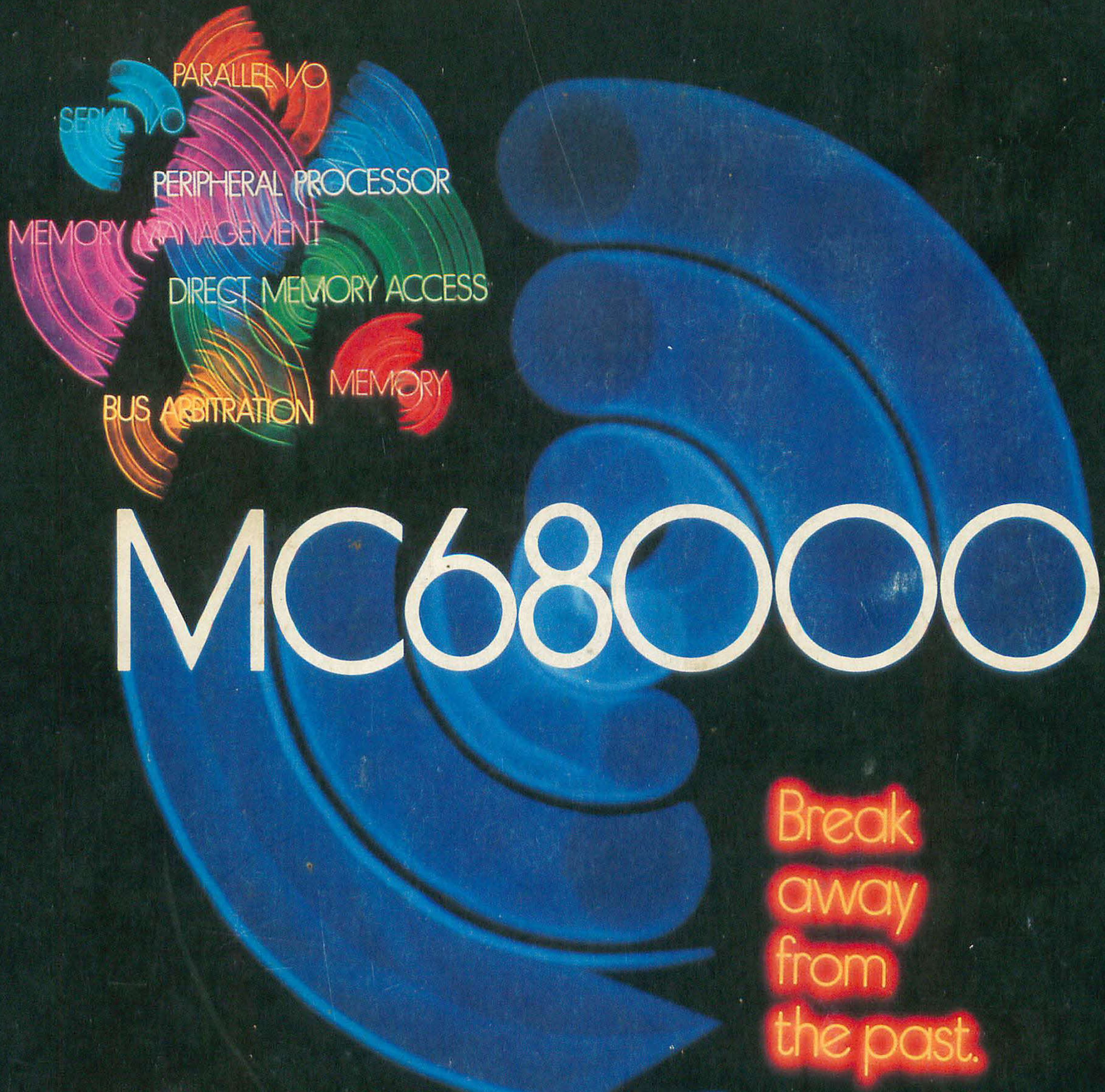


MC68000

16ビット マイクロプロセッサ
ユーザーズ・マニュアル

MOTOROLA

MOTOROLA

MC68000

16ビット マイクロプロセッサ
ユーザーズ・マニュアル

IMPORTANT NOTICE

This manual provides complete documentation for the **MC68000** microprocessor. Engineering prototype microprocessors are designated **XC68000**. Differences between the MC68000 and the XC68000 are covered in the XC68000 Exception Sheet accompanying the device.

This manual is limited to information on the MC68000 **microprocessor**. It does not define a specific assembly programming language.
The language and notation used within this manual is a register transfer language explained in Appendix B.

PRELIMINARY Original Issue: September 1979
Second Edition: January 1980

序　　文

大古の昔から“より高度のもの”を求め続けてきた人類の欲望は，現代文明の中でもとどまることを知りません。一方，これに応えてきた人類の叡知もまた無限のものです。この困果関係の成果を最も身近に感じるものの一つに，エレクトロニクス機器があります。

一般家庭用器具から産業用機器まで幅広い分野をカバーするエレクトロニクス産業は，日々の技術革新を要求されております。高度化する顧客のニーズを満たすために，内部は一層複雑化の一途をたどる反面，開発時間の短縮を余儀なくされております。これらの問題の解決の一つの手段としてマイクロコンピュータの使用をあげることができます。

いまや“産業界の米”といわれている半導体集積回路(IC)は，エレクトロニクス技術革新の中心として急速な進歩をとげてきました。約10年前に生まれたマイクロコンピュータは従来のシステムと部品という関係を打破した画期的な製品です。このマイクロコンピュータも4ビットから8ビットへと進歩をとげ，いま16ビットの世界が開けつつあります。

モトローラが，システム・エンジニアの夢を一片のシリコン上に形成した製品，これが“MC68000”16ビット・マイクロコンピュータです。MC68000は，モトローラの持っているシステム技術とプロセス技術の永年の経験に最新の技術を付加して初めて完成した超LSI製品です。

この和文マニュアルがMC68000と共に皆様のお役に立つことを願っております。

昭和56年5月

目　　次

付録B 命令セット詳細 97

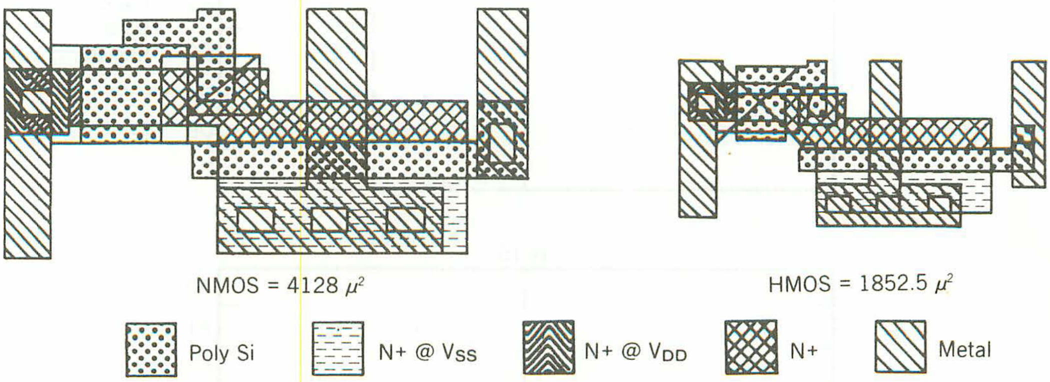
第1章　概　　論

1.1　MC 68000 マイクロプロセッサの概要

今日，半導体技術の進歩により，従来のプロセッサと比べ性能および回路の集積度が少なくとも1桁以上優れたマイクロプロセッサを量産することが可能となりました．MC 68000 は，モトローラが開発した超 LSI マイクロプロセッサ・ファミリの中核となるものです．これは，最新の半導体技術と優れた回路設計技術を融合したコンピュータ技術により完成し，最も進んだアーキテクチャをもつ16ビット・マイクロプロセッサで，8ビット・マイクロプロセッサ MC 6800 と比較すると，チップの集積度が1桁以上も優れています．

ドライ・プラズマ・エッチング，プロジェクション・プリンティングおよび HMOS（high-density short-channel MOS）回路設計技術の進歩（図1-1）は，システム・エンジニア，コンピュータ・エンジニアおよびマーケティング・エンジニアそれぞれの要求を満たす大きな技術基盤を与えました．このような技術基盤をマイクロプロセッサに応用したことにより使いやすく，しかも信頼性の高い，アプリケーションに柔軟性をもった，高性能な製品を生産することができました．

図 1-1　HMOS 回路設計技術



- Speed-power product four times better than standard NMOS
- Circuit densities twice standard NMOS

NMOS ≈ 4 PICOJOULES
HMOS ≈ 1 PICOJOULE

使用できる MC 68000 のリソースは以下のとおりです．

・32ビット長のデータ・レジスタとアドレス・レジスタ
・16 M バイト直接アドレッシング・レンジ
・56種類の強力な基本命令型式
・5種類のデータ型式
・メモリ・マップド I/O
・14種類のアドレッシング・モード

とくに，アーキテクチャを系統づけるために，レジスタ，命令（アドレッシング・モードを含む）およびデータ型式

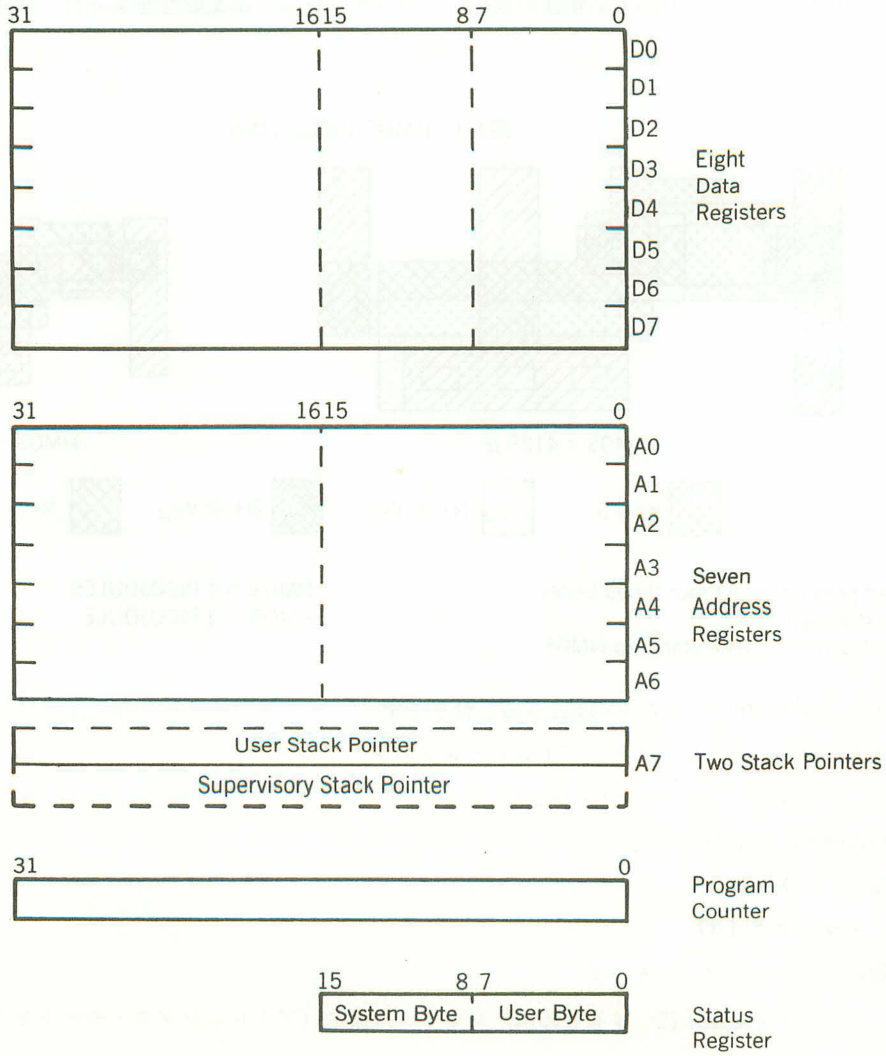
について注意を払って設計しました．系統づけられた一貫した構造を持つアーキテクチャは学びやすく，プログラムを容易にしました．これによりプログラムの記述にかかる時間とプログラム量をともに減少させることができ，ソフトウェア開発におけるコストとリスクを大幅に軽減させることが可能となりました．

システムの高いスループット（2×10^6/秒まで）の命令処理，またはデータ転送では，従来の比較的アクセス時間の遅い標準メモリを使っても達成できます．このプロセッサでは，データ・バス設計がフレキシブルであるため，低速と高速のメモリや周辺デバイスとが混在してシステムを構成しても，それぞれのアクセスごとに転送速度を自動的に最適化させ，もっとも効率的なシステム動作を行わせます．

CPU のハードウェア設計はソフトウェア技術の進歩に大きく影響されます．新世代の16ビット/32ビット・マイクロプロセッサにおいても，高級言語のコンパイラは，高級言語からコードを作るのと同じように効率良く走らせなければなりません．

MC 68000 では，一貫した構造を持つアーキテクチャ，多数のレジスタとスタック，広いアドレス・レンジ，および高級言語指向命令（LINK，UNLK，CHK 等）によって高級言語をサポートしています．オペレーティング・システム（OS）も特権命令，メモリ管理，ベクタ多重レベル割込み，トラップ構造および特定の命令（MOVEP，MOVEM，TRAP 命令等）によりサポートします．また，マルチプロセッサ・システムとして働くため，ハードウェア・インタロックとソフトウェア・インタロックの両方を備えています．

図1-2　プログラミング・モデル



MC 68000 は，共用バスおよび共用メモリ（他のプロセッサや DMA デバイスなどと共用）のためのバス・アービトレーション・ロジックを持っています．マルチプロセッサ・システムはソフト命令（TAS: test and set）によってもサポートされます．これらの特徴から MC 68000 は，ワンチップのマイクロプロセッサによるマルチプロセッサ・システムとして最大限のフレキシビリティをもっています．

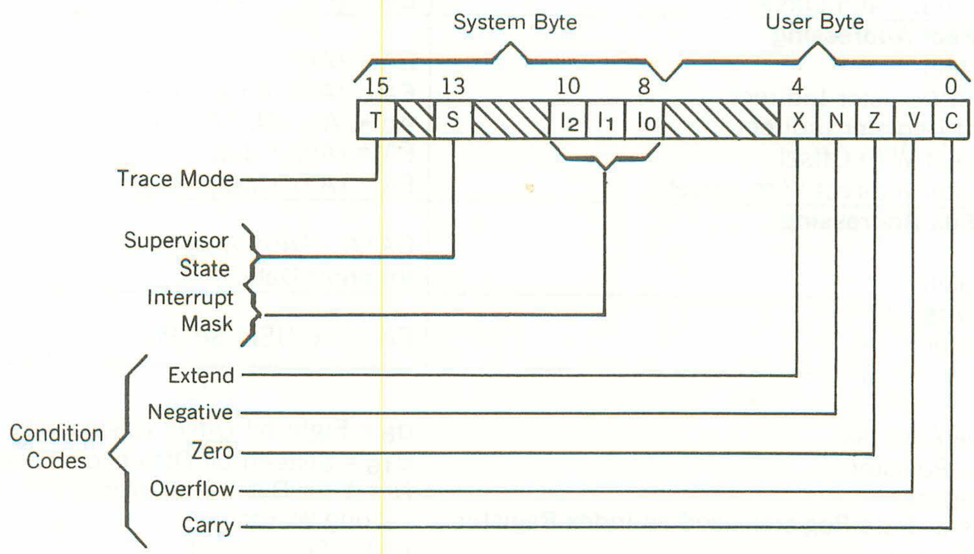
先進的なアーキテクチャのプロセッサは，大規模で複雑なジョブを効率良く処理するだけでなく，小規模で簡単なジョブに対しても同様に処理できなければなりません．MC 68000 は小規模で簡単なジョブでも大規模で複雑なジョブでもそれぞれ効率良く処理できるように設計されています．

MC 68000 は17個の32ビット・レジスタに加えて，32ビットのプログラム・カウンタと16ビットのステータス・レジスタを備えています（図1-2）．最初の8個のレジスタ（D0～D7）はデータ・レジスタでバイト（8ビット），ワード（16ビット）およびロング・ワード（32ビット）のデータ・オペレーションのために使われます．次の7個のレジスタ（A0～A6）とシステム・スタック・ポインタ（A7）はベース・アドレス・レジスタやソフトウェア・スタック・ポインタとして使用できます．また，これらのレジスタはワードやロング・ワードのアドレス・オペレーション用としても使用できます．さらに，これら17個のレジスタはインデックス・レジスタとしても使用できます．

24ビットのアドレス・バスは，16Mバイト（実際には16,777,216バイト）以上のメモリ・アドレス・レンジをもっています．この強力なアドレッシング能力はメモリ管理ユニットと相まって，大規模なモジュラ・プログラムを，わずらわしくかつ時間のかかるブック・キーピングやページング・ソフトウェアにたよらずに作成し動作させることができます．

ステータス・レジスタは，コンディション・コード〔オーバフロー(V)，ゼロ(Z)，ネガティブ(N)，キャリ(C)，拡張(X)〕と割込みマスク（8レベル）をもっています（図1-3）．そのほかにもプロセッサがトレース・モード(T)状態やスーパバイザ(S)状態にあることを表示する追加ステータス・ビットも準備されています．将来の MC 68000 ファミリの拡張に備え，ステータス・レジスタには十分広いスペースをとっています．

図1-3 ステータス・レジスタ



基本的なデータ型式は次の5種類があります．

・ビット（1ビット）

・BCD デジット（4ビット）

・バイト（8ビット）

・ワード（16ビット）

・ロングワード（32ビット）

さらに，メモリ・アドレスやステータス・ワード・データ等のデータ型式の操作も命令セットに用意されています．

表1-1 に示した14種類のアドレッシング・モードは次の6つの基本型で構成されています．

・レジスタ直接

・レジスタ間接

・アブソリュート（絶対）

・イミディエイト

・プログラム・カウンタ・リラティブ（PC 相対）

・インプライド（含意）

レジスタ間接のアドレッシング・モードは，ポストインクリメント，プリデクリメント，オフセットおよびインデックスのバリエーションをもっています．また，プログラム・カウンタ・リラティブ・モードもインデックスとオフセットのバリエーションをもっています．

表 1-1　アドレッシング・モード

Mode	Generation
Register Direct Addressing	
Data Register Direct	EA = Dn
Address Register Direct	EA = An
Absolute Data Addressing	
Absolute Short	EA = (Next Word)
Absolute Long	EA = (Next Two Words)
Program Counter Relative Addressing	
Relative with Offset	EA = (PC) + d_{16}
Relative with Index and Offset	EA = (PC) + (Xn) + d_8
Register Indirect Addressing	
Register Indirect	EA = (An)
Postincrement Register Indirect	EA = (An), An ← An + N
Predecrement Register Indirect	An ← An − N, EA = (An)
Register Indirect With Offset	EA = (An) + d_{16}
Indexed Register Indirect With Offset	EA = (An) + (Xn) + d_8
Immediate Data Addressing	
Immediate	DATA = Next Word(s)
Quick Immediate	Inherent Data
Implied Addressing	
Implied Register	EA = SR, USP, SP, PC

NOTES:

EA = Effective Address
An = Address Register
Dn = Data Register
Xn = Address or Data Register used as Index Register
SR = Status Register
PC = Program Counter
d_8 = Eight-bit Offset (displacement)
d_{16} = Sixteen-bit Offset (displacement)
N = 1 for Byte, 2 for Words and 4 for Long Words
(　) = Contents of
← = Replaces

MC 68000 の命令セットを表 1-2 に示します．これらの命令のバリエーションやサブセットがいくつかあり，これを表 1-3 に示します．プログラミングを容易にするために，特に命令セットは高級言語をサポートできるように設計

表1-2 命 令 セ ッ ト

Mnemonic	Description
ABCD	Add Decimal with Extend
ADD	Add
AND	Logical And
ASL	Arithmetic Shift Left
ASR	Arithmetic Shift Right
B_{CC}	Branch Conditionally
BCHG	Bit Test and Change
BCLR	Bit Test and Clear
BRA	Branch Always
BSET	Bit Test and Set
BSR	Branch to Subroutine
BTST	Bit Test
CHK	Check Register Against Bounds
CLR	Clear Operand
CMP	Compare
DB_{CC}	Test Cond., Decrement and Branch
DIVS	Signed Divide
DIVU	Unsigned Divide
EOR	Exclusive Or
EXG	Exchange Registers
EXT	Sign Extend
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link Stack
LSL	Logical Shift Left
LSR	Logical Shift Right
MOVE	Move
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral Data
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG	Negate
NOP	No Operation
NOT	One's Complement
OR	Logical Or
PEA	Push Effective Address
RESET	Reset External Devices
ROL	Rotate Left without Extend
ROR	Rotate Right without Extend
ROXL	Rotate Left with Extend
ROXR	Rotate Right with Extend
RTE	Return from Exception
RTR	Return and Restore
RTS	Return from Subroutine

表1-2 命令セット(つづき)

Mnemonic	Description
SBCD	Subtract Decimal with Extend
Scc	Set Conditional
STOP	Stop
SUB	Subtract
SWAP	Swap Data Register Halves
TAS	Test and Set Operand
TRAP	Trap
TRAPV	Trap on Overflow
TST	Test
UNLK	Unlink

表1-3 命令タイプの変形

Instruction Type	Variation	Description
ADD	**ADD**	Add
	ADDA	Add Address
	ADDQ	Add Quick
	ADDI	Add Immediate
	ADDX	Add with Extend
AND	**AND**	Logical And
	ANDI	And Immediate
CMP	**CMP**	Compare
	CMPA	Compare Address
	CMPM	Compare Memory
	CMPI	Compare Immediate
EOR	**EOR**	Exclusive Or
	EORI	Exclusive Or Immediate
MOVE	**MOVE**	Move
	MOVEA	Move Address
	MOVEQ	Move Quick
	MOVE from SR	Move from Status Register
	MOVE to SR	Move to Status Register
	MOVE to CCR	Move to Condition Codes
	MOVE to USP	Move to User Stack Pointer
NEG	**NEG**	Negate
	NEGX	Negate with Extend
OR	**OR**	Logical Or
	ORI	Or Immediate
SUB	**SUB**	Subtract
	SUBA	Subtract Address
	SUBI	Subtract Immediate
	SUBQ	Subtract Quick
	SUBX	Subtract with Extend

されています．それぞれの命令は，若干の例外を除いてバイト，ワード，ロング・ワードの操作ができ，ほとんどの命令は14種類のアドレッシング・モードを自由に使用できます．これらの命令型式，データ型式，アドレッシング・モードを組み合わせると1,000種類以上の命令が使えます．これらの命令には符号付きおよび符号なし乗算と除算，

クイック（quick）演算操作，BCD演算および拡張処理（トラップ命令を介する）が含まれます．また対称性の高い専用のマイクロコード構造により，将来の拡張に対して堅実で，フレキシブルに対応ができます．

1.2　ソフトウェアの特徴

MC 68000は，プログラムをより簡単に，より早く，そしてより信頼性を高めるために多くの特徴を備えています．

（1）　系統化されたアーキテクチャ

MC 68000のアーキテクチャは高度に系統化されているので，高級言語で記述するのと同様な容易さでアセンブラ言語によりプログラムを記述できます．レジスタやメモリでの整数データの操作はいろいろなデータ型式が使用できます．バイト（8ビット），ワード（16ビット），ロング・ワード（32ビット）の整数データの操作のために，それぞれ別な命令を必要としません．プログラマはそれぞれのオペレーション・タイプごとに1つのニーモニックを覚えるだけでよく，それにデータ・サイズ，ソース・アドレッシング・モードおよび行き先アドレッシング・モードを記述すればよいのです．これにより命令数を減らすことができました．

多くの命令は2オペランド型式をもち，これはMC 68000のフレキシビリティと能力を大幅に高めました．整数データのオペランド操作では，データ・レジスタとメモリ・ロケーションは，ソースにも行き先にも自由に指定することができます．

アドレッシング・モードは，効率を損わない限りシンプルにしてあり，命令操作（ニーモニック）からも独立しています．さらにつけ加えるならば，すべてのアドレス・レジスタをレジスタ直接，レジスタ間接およびインデックス・アドレッシング・モードに使用できます．ただし定義によりイミディエイト，アドレッシング・モード，プログラム・カウンタ・リラティブ・アドレッシング・モードおよびアブソリュート・アドレッシング・モードはアドレス・レジスタを使用しません．また，どのアドレス・レジスタあるいはデータ・レジスタもインデックス・レジスタとして使用できます．8本のアドレス・レジスタのいずれもがレジスタ間接ポストインクリメント／プリデクリメント・アドレッシング・モードを使用することによりプログラム・スタック・ポインタとしてスタック・オペレーションに使用できます．このようにアドレス・レジスタも系統化されています．ただし，アドレス・レジスタA 7は特別のレジスタです．A 7はその通常のアドレッシング能力に加え，スーパバイザ状態でシステム・スタック・ポインタとして働き，トラップや割込み時のプログラム・カウンタとステータス・レジスタのスタックに役立ちます．

（2）　構造化モジュラ・プログラム

ここ数年間に，マイクロプロセッサのプログラム技術は急速に発展し，プログラム作成をより容易に，かつ信頼できるものにするための多くの技術が開発されました．これらの技術は，たとえば，モジュラ・プログラム技術のように定められたプログラミング構造を遵守するよう，記述時の規則が定められています．モジュラ・プログラミングでは要求される機能ないし処理を，容易にプログラムやデバッグのできる小単位のモジュールやサブルーチンに分割しなければなりません．このような問題は構造化アセンブラやPASCALのようなブロック構造高級言語を使用することで容易に解決できます．しかし，このような概念は，再入および再帰を基本動作とするモジュール間のパラメータを容易にやりとりできない限り，利用することは容易ではありません．（再入ルーチンは呼び出しにおいてのプログラムのデータを破壊することなく動作できなければなりません．再帰ルーチンは自分自身を呼び出して使用できるものです．）MC 68000は再入形式のモジュラ・プログラムを作りやすいアーキテクチャをもっています．2つの相

補する命令，LINK と UNLK 命令ではスタック上のデータ・エリアのリンクされたリストを操作することにより，サブルーチン・コールのオーバヘッドを減少させます．MOVEM（move multiple registers）命令においてもサブルーチン・コールのオーバヘッドを減少させます．MOVEM 命令によって，プログラムをが指定するレジスタ群をアドレスを指定するだけで一度に転送できます．TRAP 命令には 16 個のソフトウェア・トラップ・ベクタがあり，システム・コール・ルーチンやユーザのマイクロ・ルーチンを処理するのに有効です．また，その他の構造化プログラミング技法をサポートする命令としては JSR，BSR，RTS 命令はもちろんのこと PEA（push effective address），LEA（load effective address），RTR（return and restore），RTE（return from exception）命令があります．

マイクロプロセッサの強力なベクタ割込み技能を使用することにより，再入モジュラ入出力ルーチンを直接作成することができます．192 のベクタ・ロケーションと 7 つのオート・ベクタ・ロケーションをもつ 7 レベルのマスク可能な優先割込み機能によりフレキシブルな I/O コントロールができます（総計 255 のベクタ・ロケーションがあり，これを割込み，ハードウェア・トラップ，ソフトウェア・トラップに割り当てています．）

（3） エラー検出機能

MC 68000 は，エラーの発生を減少させるための特徴をいくつか備えていますが，そのうち系統だったアーキテクチャと構造化モジュラ・プログラミング機能についてはすでに説明しました．

システム・プログラマにとって非常に重要視される機能は，プログラム・エラーと虫（bug）を検出する機能です．MC 68000 のハードウェア・トラップは下記のエラー状態となった時に，内部状態に異常があったことを通知します．

・奇数アドレスでのワード・アクセス
・違法命令
・オペコード・エラー
・違法メモリ・アクセス（バス・エラー）
・ゼロによる除算
・オーバ・フロー・コンディション・コード（TRAPV 命令）
・レジスタの限界オーバ（CHK 命令）
・擬似割込み

そのうえ，プログラマは16個のソフトウェア TRAP 命令を使用して，エラー検出および訂正ルーチンを自由に作成することができます．

他のエラー検出ツールとして配列（array）の限界をチェックする CHK 命令（check register against bounds）があります．これはデータ・レジスタに妥当な記載がされているかを確認する命令です．これにより，レジスタの容量が負になるか，限界を越えた場合にトラップします．

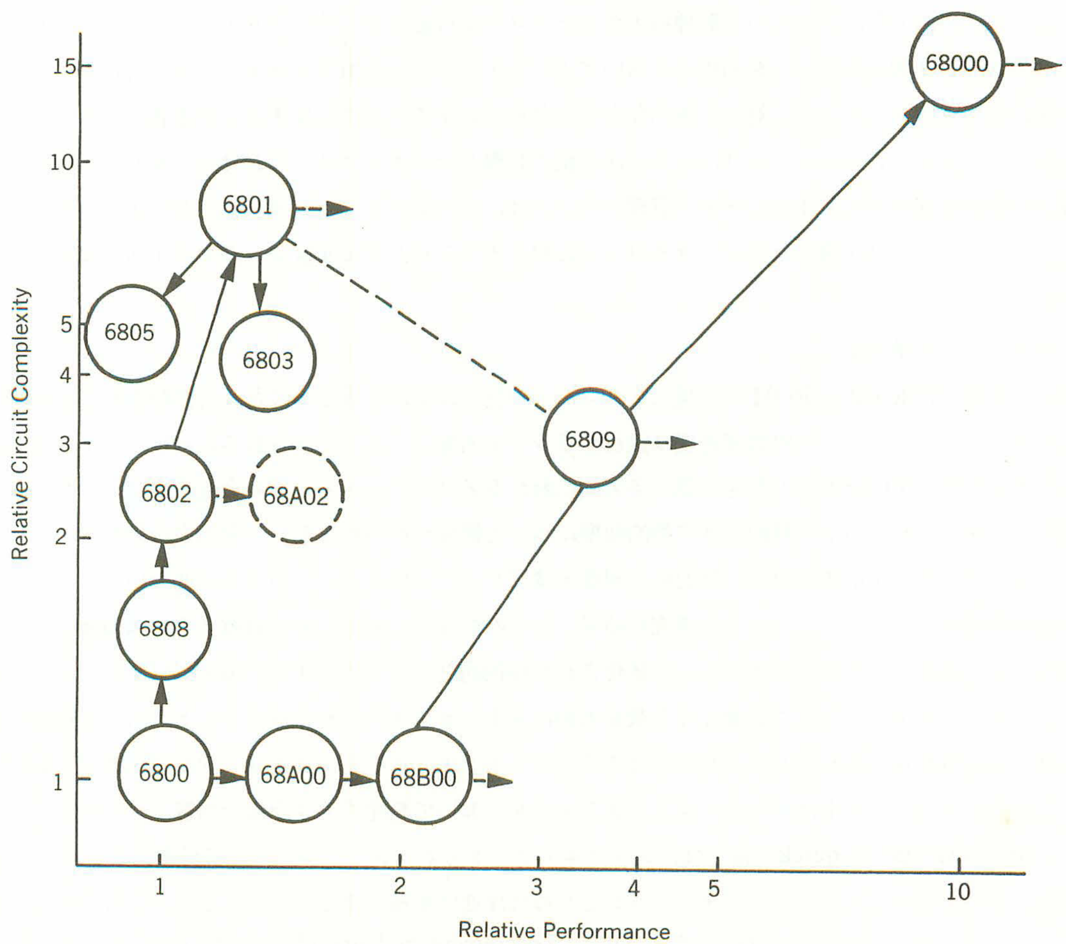
最後に MC 68000 は，一つ一つの命令ごとにプログラムのデバッグを行うトレース機能を備えています．このトレース・モードはそれぞれの命令実行後でトラップした時，トレース・ルーチンに入るようになっています．プログラマはまるでマイクロプロセッサをスーパバイザ状態にしているのと同じような状態で利用できます．また外部メモリ管理デバイスを使用することによりメモリ選択エリアの記憶保護が行えます．これによりスーパバイザ・モードとユーザ・モードのエラー・プロテクション機能を強化することができます．

(4) 将来のフレキシビリティ

マイクロプロセッサ超LSI回路技術は，かつてない速度で進歩しています．たとえば，モトローラのMC 6800ですが，オリジナルは1974年に発表されましたが，さらにいくつかの進んだ製品に発展しました．図1-4に示すように，この発展は2つの方向をたどりました．一方は，機能面が強化されたMC6802とMC6801マイクロコンピュータであり，もう一方は効率面が強化されたMC68A00，MC68B00とMC6809マイクロプロセッサです．オリジナルのMC 6800でのアーキテクチャの基盤があったので，NMOS，超LSIのスピードおよび集積度を高めることにより，これらの改良品の開発ができました．また，これらの開発ではオリジナルのMC 6800マイクロプロセッサとのコンパチビリティおよび一貫性が保たれています．

MC 68000は独特のマルチレベル・マイクロプログラム構造を持ち，命令の実行には融通性があります．実際，命令のオペコード・マップの1/8以上が将来の命令追加のために特別に確保されています．その間に，ユーザは現在の命令セットにはない命令を，トラップ命令とエミュレータ・トラップを用いて作成し実行することができます．

図1-4 モトローラ・マイクロプロセッサの進歩



(5) メモリ管理

半導体メモリの価格が下がり続けているので，高級言語や高度なディスク・オペレーティング・システムを容易に作成し使用できるようになりました．モトローラの新製品であるMC 68000は，複雑なメモリを使用するアプリケーションに対し効率よくコントロールすることを可能にしました．このような用途に応えるために，MC 68000では16 Mバイトのメモリを直接アクセスできます．その命令によってオペランド・サイズを記述することにより，この

大アドレス空間は，ワードまたはバイト・ベースで直接アクセスされ，効率良く管理されます．上位データ・ストローブ（$\overline{UDS}$）と下位データ・ストローブ（$\overline{LDS}$）を使って，上位バイト，下位バイトあるいはワードを容易にアクセスできます．

プログラマが効率良くメモリを利用できるように，さらにいくつかの便利な特徴があります．レジスタ間接，インデックス，ショートまたはロング・アブソリュートおよびプログラム・カウンタ・リラティブなどの強力なアドレッシング・モードによって，メモリへのアクセスが整然と行えます．これらのアドレッシング・モードは，アドレス計算（レジスタ間接とインデックス），メモリへの直接アドレス（ショートおよびロング・アブソリュート）と，ポジション・インデペンドないし再配置コーディング（プログラム・カウンタ・リラティブ）を容易にします．もちろん，プリデクリメント/ポストインクリメント・レジスタ間接アドレッシング・モードを利用することによってプログラマは8個のスタックや待ち行列を同時に作成して効率良くメモリ内のデータ管理を行えます．また，プログラマがソフトウェアでメモリ管理を行えるようにCHK命令を用意してあります．この命令によって，メモリ保護／管理をソフトウェアで実現することができます．

MC 68000 のもう一つの特長はユーザ状態とスーパバイザ状態の区別があることです．スーパバイザ状態では，プロセッサ・システム内で保護されている数種のオペレーションが可能です．特にプロセッサがユーザ状態の場合，プログラマは外部メモリ管理ユニットを利用して広いアドレス・レンジを使用できます．メモリ管理オペレーションはユーザ状態においてはプログラマに対して明白なものとなっています．また，変更および更新はスーパバイザ状態でのみ行われます．メモリ管理ユニットは，メモリ区分化の管理とマルチ・タスクにおけるメモリ・リロケーションとメモリ保護の動的管理を提供します．メモリ管理ユニットは，リード・オンリ・データ，リード／ライト・データ，プログラム・コードおよび保護されたデータやコードに割り当てられたメモリ区分へのアクセスを規則的にします．

(6) 命令コードと処理速度

低価格，高集積度のRAMとROMの出現によりコードの長さはもはや重要ではないと誤解されているかも知れません．しかしマイクロプロセッサの処理速度は実行するコード密度に大きく影響されるため，コードは徹底的に研究され決定されました．MC 68000 の開発初期に多方面にわたるアプリケーションでの命令や命令シーケンスの使用について徹底的に調査され，命令の静的頻度や動的頻度について研究が行われました．動的命令頻度とはどれだけの頻度で命令が実行されたかを示すもので，静的命令頻度とはプログラムのリスト中の命令頻度ないしアセンブラが命令を検出する頻度で測られます．このような研究の結果，命令の処理能力向上に主に貢献するのは高度に系統化されたアーキテクチャであることがわかりました．系統化された規則的なアーキテクチャ，命令セットおよびアドレッシング・モードは一定の仕事をするのに必要な命令数を大幅に減少させます．これら命令シーケンスと処理速度を改善するために多くの命令が加えられました．たとえばクイック・イミディエイト・アドレッシング・モードを用いたシングル・ワードの加算，減算命令はデータ・レジスタとメモリにおいて高速で小さな値の演算オペレーションを可能にしました．MOVEQ (move quick) 命令は，シングル・ワード・オペレーションで符号付きバイト・データを任意のレジスタにロードできます．ループ・オペレーションの処理速度を速くするために，1つの命令でコンディションのテストを行い，デクリメントし分岐できる DB_{cc}（decrement and branch）命令が備わっています．もちろん TRAP, MOVEM (move multiple registers), LINK (link stack), UNLK (unlink stack), CHK (check limit) 命令もサブルーチン・コール，オペレーティング・システム・コールやスタック・オペレーションに必要なコードを大幅に減少できます．

コードを減らし，演算オペレーションの性能を高める命令として，MULS(signed multiply)と MULU(unsigned multiply)，および DIVS (signed divide) と DIVU (unsigned divide) があります．BCD 演算オペレーション

用としてABCD, SBCDがあります．データの移動ないし転送の向上のため，バイト， ワード，ロング・ワードの転送をすべてのデータ・アドレッシング・モードで行える強力なMOVE命令が用意されています．すべてのデータ・アドレッシング・モードが使用できるのでレジスターレジスタ，レジスターメモリ，メモリーレジスタ，メモリーメモリ間のデータ転送ができます．

プロセッサの処理能力を向上させる強力な命令体系に加え，アーキテクチャ上の数多くの特徴によりプログラムの実行時間を大幅に減少させることができます．マルチプレクスされていない個々に独立しているアドレス・バスとデータ・バス，命令先取り機能，32ビットの内部レジスタ群によってMC 68000の性能は他に類をみないものになっています．

(7) ソフトウェア・サポートと MC 6800 とのコンパチビリティ

システム設計者とプログラマは， MC 68000を使用したアプリケーションに携わる場合，ハードウェアとソフトウェアの完全にコンパチブルな開発支援ツールを使用できます．MC 68000 は，ディスク・オペレーティング・システム，デバッグ・エイド，アセンブラおよび高級言語を含む広範囲のソフトウェアを持つ開発支援ツールでサポートされます．

この新しいマイクロプロセッサは注意深い設計がされていますので， MC 6800 よりも多く， かつ大きなレジスタおよび一貫したアーキテクチャとフレキシブルなアドレッシング・モードを付加することにより， MC 6800 に取って代わる強力な命令セットをもっています．これにより現存する MC 6800 のプログラムを効率よく移植できます．

MC 68000 と MC 6800 間の両立性については，ソフトウェアだけの両立性にとどまらず，ペリフェラル・コントローラとのインターフェースについても考慮されています．ですからモトローラの インテリジェントなMC 6800ペリフェラル・デバイス群〔MC 6854 : Advanced Data Link Controller (ADLC), MC 68488 : General Purpose Interface Adapter(GPIA)等〕は，MC 68000と直接インターフェースできます．そのため3本の信号線〔E(enable), $\overline{\text{VMA}}$(valid memory address)，および $\overline{\text{VPA}}$(valid peripheral address)〕などがモトローラの標準 MC 6800 ペリフェラル・デバイス群と直接インターフェースをするために設けられています．

第2章　データ構成とアドレッシング・モード

本章では，MC68000 のデータ構成およびアドレッシング・モードについて説明します．

2.1　オペランド・サイズ

オペランド・サイズは，次のように定義します．

バイト　　　　＝8ビット

ワード　　　　＝16ビット

ロング・ワード＝32ビット

各命令のオペランド・サイズは，オペコードの後に指定する場合と，指定しなくてもよい場合とがあります．オペランド・サイズを指定する命令では，バイト（B)，ワード（W)，ロング・ワード（L）で指定します．オペランド・サイズを指定しない命令では，バイト（B)，ワード（W)，ロング・ワード（L）のいずれかにサイズが決まっています．

2.2　レジスタのデータ構成

8個のデータ・レジスタは，1，8，16，そして32ビットのデータ・オペランドをサポートします．7個のアドレス・レジスタは，アクティブ・スタック・ポインタとともに，32ビットのアドレス・オペランドをサポートします．

(1)　データ・レジスタ

データ・レジスタは，32ビット構成で，バイト・オペランドは下位8ビット，ワード・オペランドは下位16ビット，そしてロング・ワード・オペランドは全32ビットを使います．最下位ビットはビット0，最上位ビットは，ビット31として定義されています．

データ・レジスタは，下位ビット（バイトまたはワード）をソース・オペランド，あるいは行き先オペランドとして使用する場合，該当する部分のみ変化し，残りの上位ビットは使用しません．

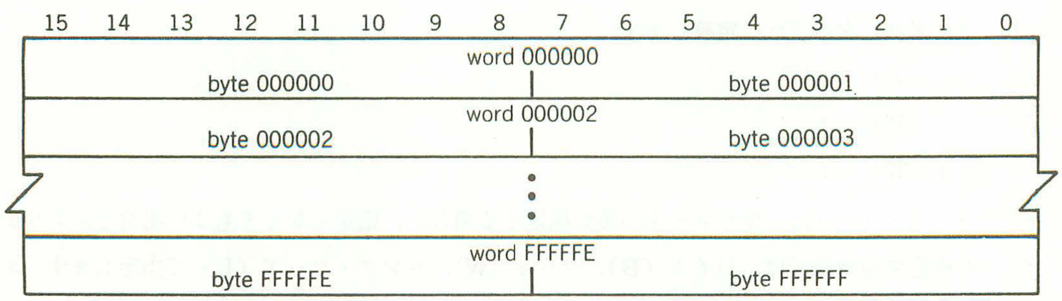
(2)　アドレス・レジスタ

アドレス・レジスタとスタック・ポインタは，32ビットの幅を持ち，32ビット長のアドレスを操作できます．アドレス・レジスタでは，バイト・サイズ（8ビット）のオペランドは使用できません．このため，アドレス・レジスタをソース・オペランドとして使用する場合は，操作するデータ・サイズに応じてワード・オペランド（16ビット)，ロング・ワード・オペランド（32ビット）のいずれかを使用します．アドレス・レジスタを行き先オペランドとして使用する場合は，操作のデータ・サイズにかかわらずレジスタ全体（32ビット）が影響を受けます．操作するデータ・サイズがワード（16ビット）の場合，操作の実行に先立って32ビットに拡張されます．

2.3 メモリ内のデータ構成

上位バイトは図2-1に示したワードと同様に偶数アドレスでアドレスされます．下位バイトのアドレスは，偶数のワード・アドレスより，1アドレス増加した奇数アドレスとなります．命令およびマルチ・バイト・データは，ワード単位（偶数バイト）でのみアクセスされます．ロング・ワード・データがアドレスn（nは偶数）に格納されている場合，そのデータの下位のワードはアドレスn+2に格納されています．

図2-1　メモリ内のワード構成



MC 68000の支援するデータには，1，8，16，および32ビットの整数データ，32ビットのアドレス・データおよびBCDデータがあります．これらのデータ型式を図2-2に示します．

図2-2　メモリ内のデータ構成

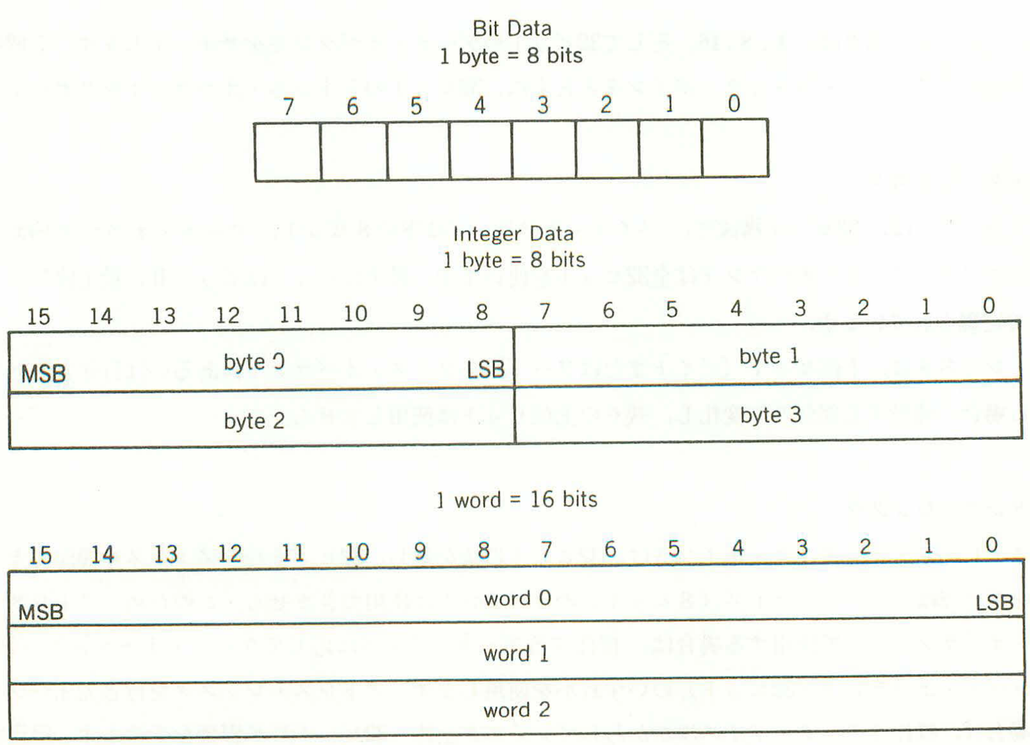
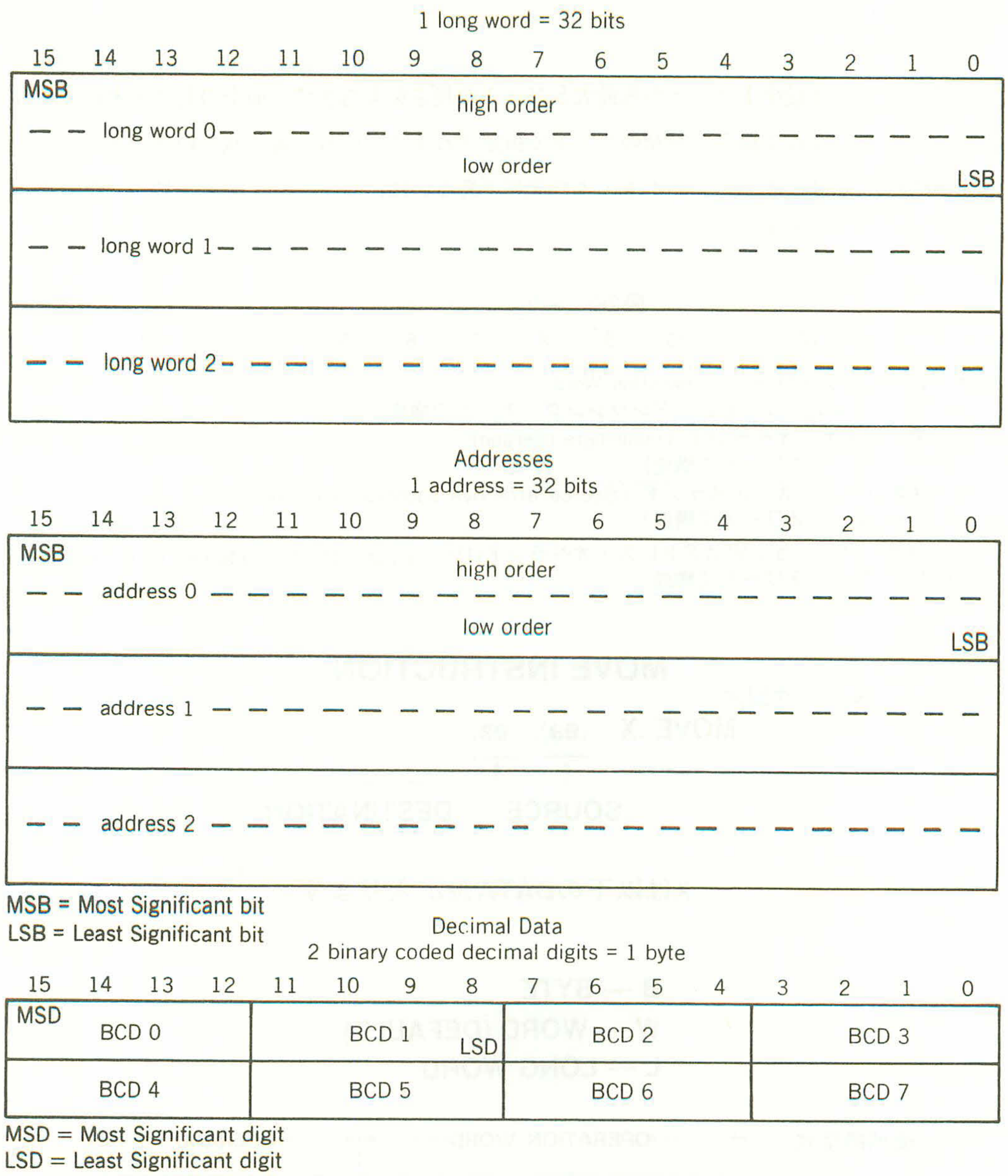


図 2-2　メモリのデータ構成



2.4 アドレッシング

MC68000 の命令は，2 種類の情報をもっています．一つは実行する機能を指定するもの，もう一つは実行するオペランドの位置を指定するものです．

オペランドの位置を指定する方法は，次の 3 通りがあります．

・レジスタ指定——命令のレジスタ・フィールドにおいて，レジスタ番号を指定します．

・実効アドレス——使用する実効アドレス・モードを指定します．

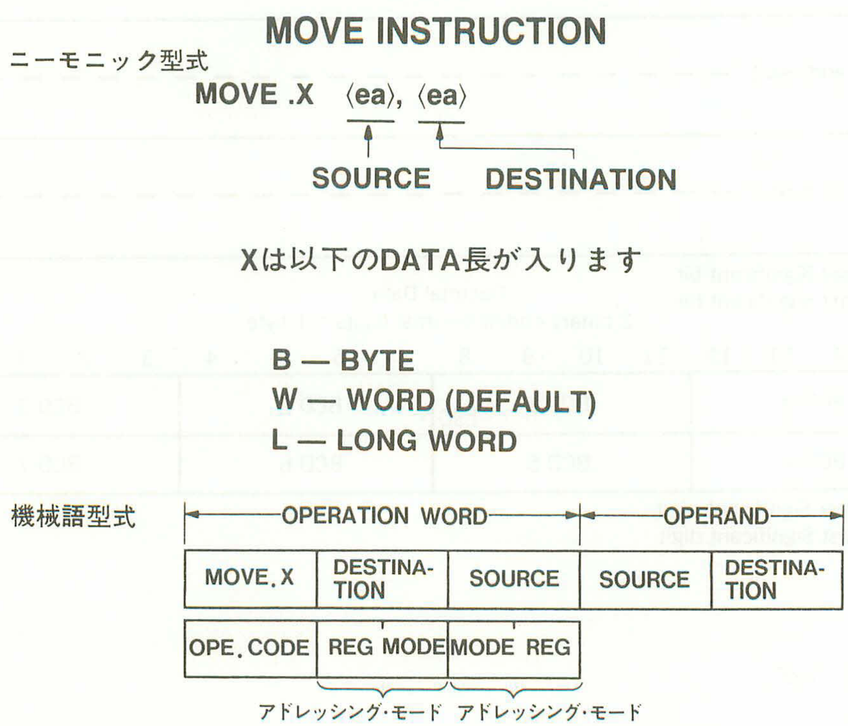
・インプリシット・レファレンス (implicit reference)——ある命令によりあらかじめ決められている特定のレジスタを指定します．

2.5 命令フォーマット

図2-3に示すように，.命令は最小1ワードから最大5ワードの長さをもちます．命令の長さと実行される操作は，オペレーション・ワードと呼ばれる命令の最初のワードで指定されます．それに続く残りのワードはオペランドを示しています．これらのワードは，イミディエイト・オペランドあるいは，オペレーション・ワードの実効アドレス・モードで指定されたオペランドです．

図2-3 命令フォーマット

15 ～ 0
オペレーション・ワード（Operation Word） （オペレーション・コードとアドレッシング・モードで構成）
イミディエイト・オペランド（Immediate operand） （1ワードないし2ワードで構成）
ソース実効アドレス・オペランド（Source effective address extension） （1ワードないし2ワードで構成）
ディスティネーション実効アドレス・オペランド（Destination effective address extension） （1ワードないし2ワードで構成）



2.6 プログラム／データ・レファレンス

MC68000のメモリ・レファレンスは，プログラム・レファレンスとデータ・レファレンスにわけられます．プログラム・レファレンスは，プログラムが格納されているメモリ領域をアクセスします．また，データ・レファレンスはデータが格納されているデータ領域をアクセスします．オペランドで指定するデータの読み出しはデータ領域から読まれ，オペランドで指定するデータの書き込みはすべてデータ領域へ書き込まれます．

2.7 レジスタ表記法

付録Bで命令操作の説明に使われるレジスタ・シンボル（RTL：Register transfer language）の定義を次に示します．

An：アドレス・レジスタ（nはレジスタ番号）

Dn：データ・レジスタ（nはレジスタ番号）

Rn：任意のアドレス・レジスタもしくは，データ・レジスタ（nはレジスタ番号）

PC：プログラム・カウンタ

SR：ステータス・レジスタ

CCR：コンディション・コード（ステータス・レジスタの下位バイト）

SP：スタック・ポインタ（ユーザ・スタック・ポインタおよびスーパバイザ・スタック・ポインタ）

USP：ユーザ・スタック・ポインタ

SSP：スーパバイザ・スタック・ポインタ

d：ディスプレースメント

N：バイト単位（1，2，4）のオペランド・サイズ

2.8 アドレス・レジスタ間接（Address Register Indirect）表記法

アドレス・レジスタで，メモリ・ロケーションを指定する場合，アドレッシング・モードは，アドレス・レジスタ間接と呼ばれます．間接（indirect）という言葉は，命令の操作によってアドレス・レジスタ自体を直接指定するのではなく，アドレス・レジスタを指定することによりそのアドレス・レジスタが指定しているメモリ・ロケーションを指定することを意味します．間接モードの RTL 表記法は，アドレス・レジスタの後に，@の記号で指定します．たとえば，A 4 @はアドレス・レジスタ 4 の内容がオペランドのメモリ・ロケーションを示しています．

2.9 レジスタ指定

一つの命令の中にあるレジスタ・フィールドは，使用するレジスタを指定します．命令内の他のフィールドでは選択したレジスタがアドレス・レジスタであるか，データ・レジスタであるか，またどのようにそのレジスタを使用するかを指定します．

2.10 実効アドレス

ほとんどの命令では，オペランドの位置はオペレーション・ワードの実効アドレス・フィールドを使用して指示します．たとえば，図 2-4 は単一実効アドレス命令オペレーションの一般的フォーマットです．実効アドレス・フィールドは，3 ビットのモード・フィールドと 3 ビットのレジスタ・フィールドからなっています．モード・フィールドはいろいろなアドレス・モードを選択します．レジスタ・フィールドはレジスタ番号を指定します．

実効アドレッシング・モードは，レジスタ直接，レジスタ間接，アブソリュート，プログラム・カウンタ・リラティブ，イミディエイトの各モードがあります．

図2-4 単一実効アドレス命令のオペレーション・ワードの一般的フォーマット

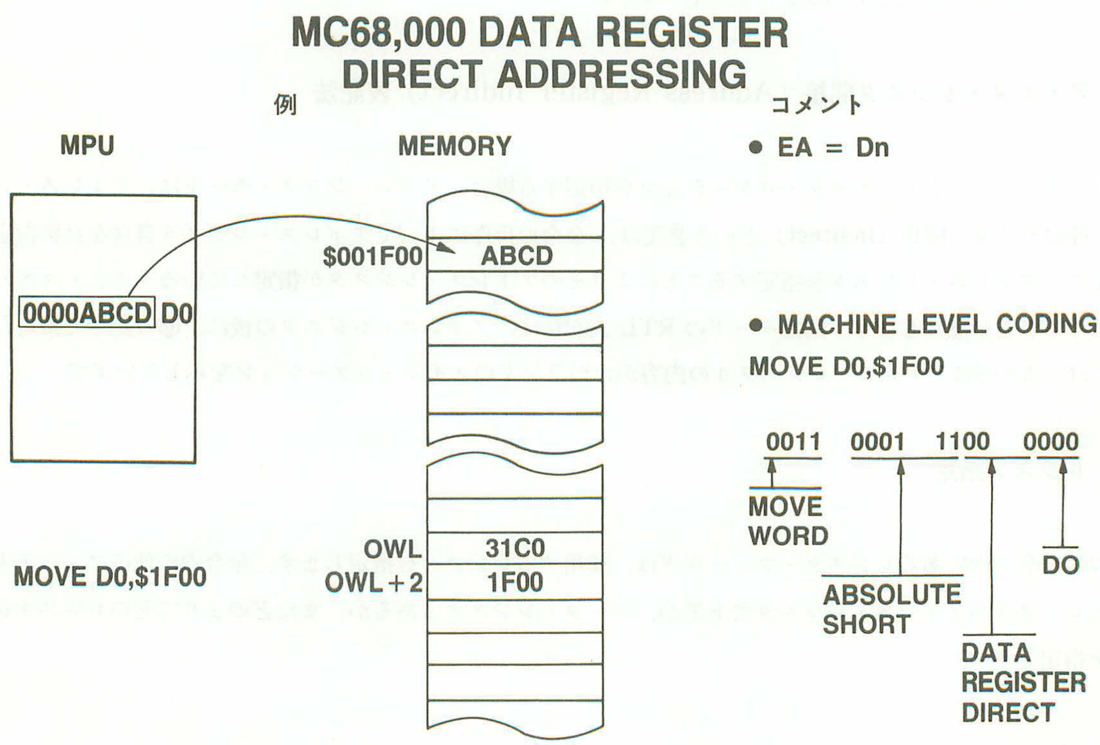
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	Effective Address					
										Mode			Register		

（1） レジスタ直接モード

実効アドレッシング・モードは，16個のマルチ・ファンクション・レジスタ（D0～D7，A0～A7）の中の1個をオペランドとして指定できます．

▩データ・レジスタ直接　オペランド・データは，実効アドレス・レジスタ・フィールドで指定したデータ・レジスタに格納されます．

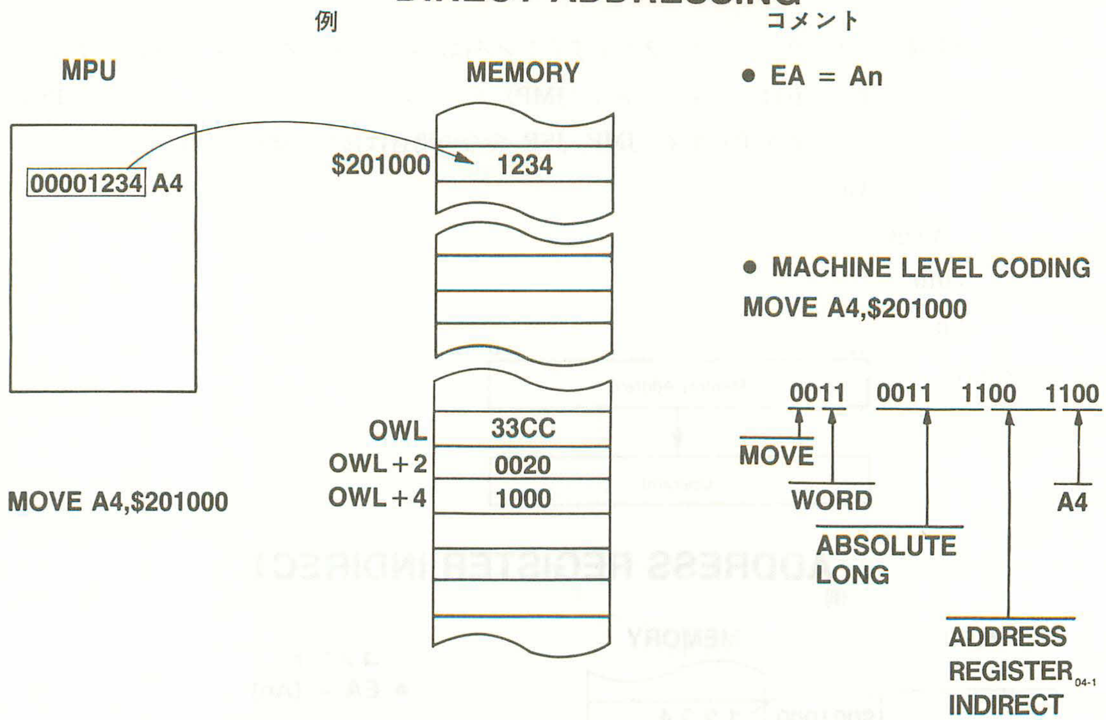
ジェネレーション　：EA＝Dn
RTL表記　：Dn
モード　：000
レジスタ　：n
データ・レジスタDn　Operand



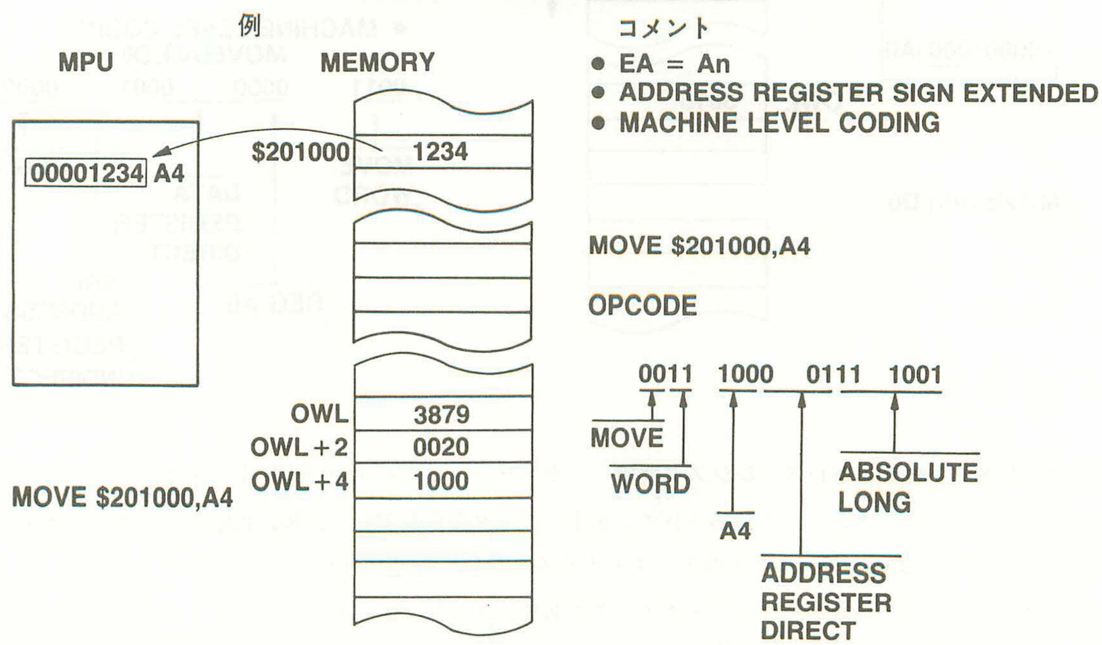
▩アドレス・レジスタ直接　オペランド・データは，実効アドレス・レジスタ・フィールドで指定したアドレスレジスタに格納されます．

ジェネレーション　：EA＝An
RTL表記　：An
モード　：001
レジスタ　：n
アドレス・レジスタAn　Operand

MC68,000 ADDRESS REGISTER DIRECT ADDRESSING



MC68,000 ADDRESS REGISTER DIRECT ADDRESSING



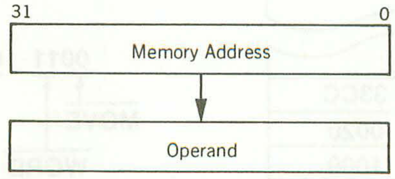
(2) メモリ・アドレス・モード

これらの実効アドレッシング・モードでは，データはメモリに格納されており，オペランドでそのデータのアドレスを指定します．

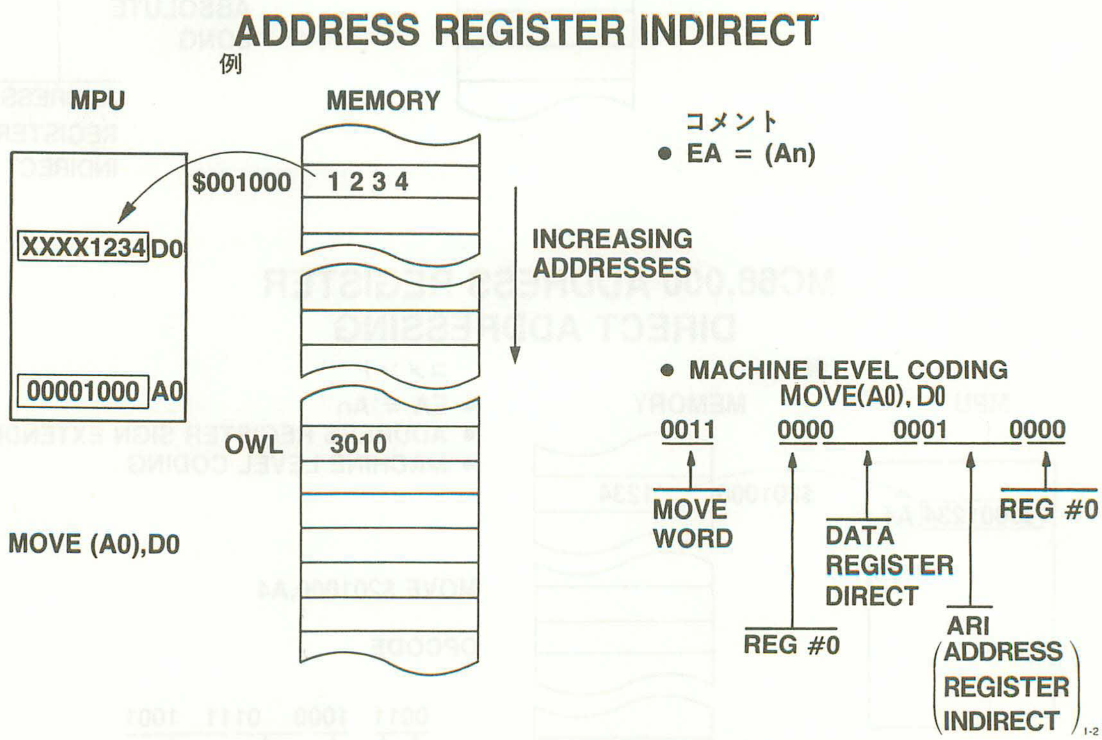
■アドレス・レジスタ間接　オペランド・データを示すアドレスは，レジスタ・フィールドで指定されるアドレス・レジスタに格納されています．これは，ジャンプ命令（JMP）とジャンプ・ツー・サブルーチン命令（JSR）を除いて，オペランドによってデータ位置を示します（JMP，JSR 命令の場合は命令位置を示します）．

ジェネレーション　：EA=(An)
RTL 表記　：An @
モード　：010
レジスタ　：n

アドレス・レジスタAn



メモリ・アドレス



■ポストインクリメント付きアドレス・レジスタ間接　オペランド・データを示すアドレスは，レジスタ・フィールドで指定されるアドレス・レジスタに格納されています．データ操作を実行した後，指定されたアドレス・レジスタでバイト，ワードまたはロング・ワードのデータ・サイズに応じ，指定されたアドレス・レジスタを1，2または4進めます．アドレス・レジスタがスタック・ポインタであり，オペランド・サイズがバイトの場合はスタック・ポインタをワード単位で保持させるため，2進させます．これは，データ・レファレンスに分類されます．

ジェネレーション　：EA=(An)
　　　　　　　　　　An=An+N
RTL 表記　：An @+

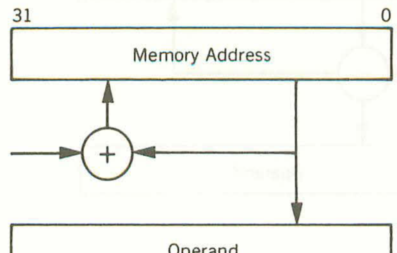
モード　　　　　　：011

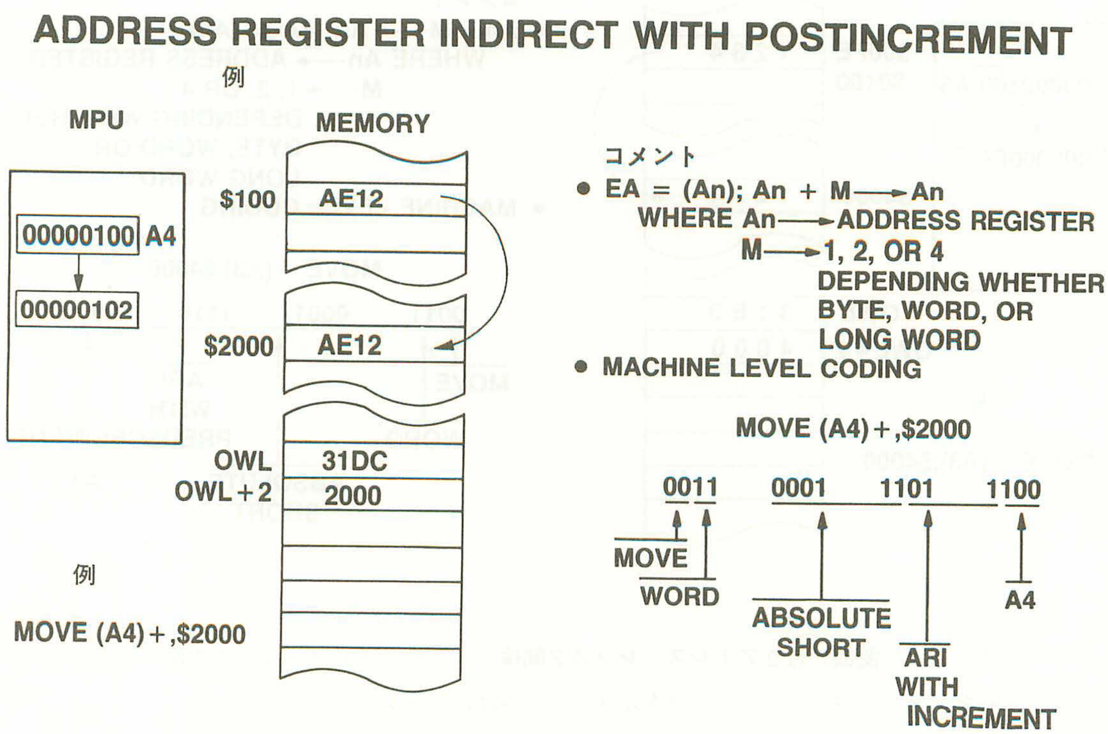
レジスタ　　　　　：n

アドレス・レジスタ

オペランドの長さ
（1，2あるいは4）

メモリ・アドレス





■プリデクリメント付きアドレス・レジスタ間接　オペランドのアドレスは，レジスタ・フィールドで指定されるアドレス・レジスタに格納されています．データ操作を実行する前に，バイト，ワード，ロング・ワードのオペランド・サイズに応じてアドレス・レジスタが1，2または4減らされます．アドレス・レジスタがスタック・ポインタであり，オペランド・サイズがバイトの場合は，スタック・ポインタをワード単位に保持するため2を減らします．これはデータ・レファレンスに分類されます．

ジェネレーション　：An=An−N

　　　　　　　　　　EA=(An)

RTL表記　　　　　：An@−

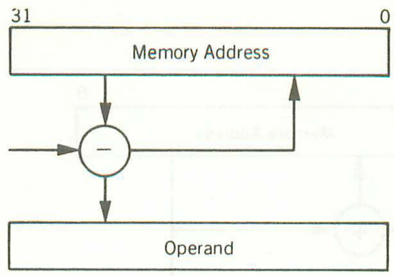
モード　　　　　　：100

レジスタ　　　　　：n

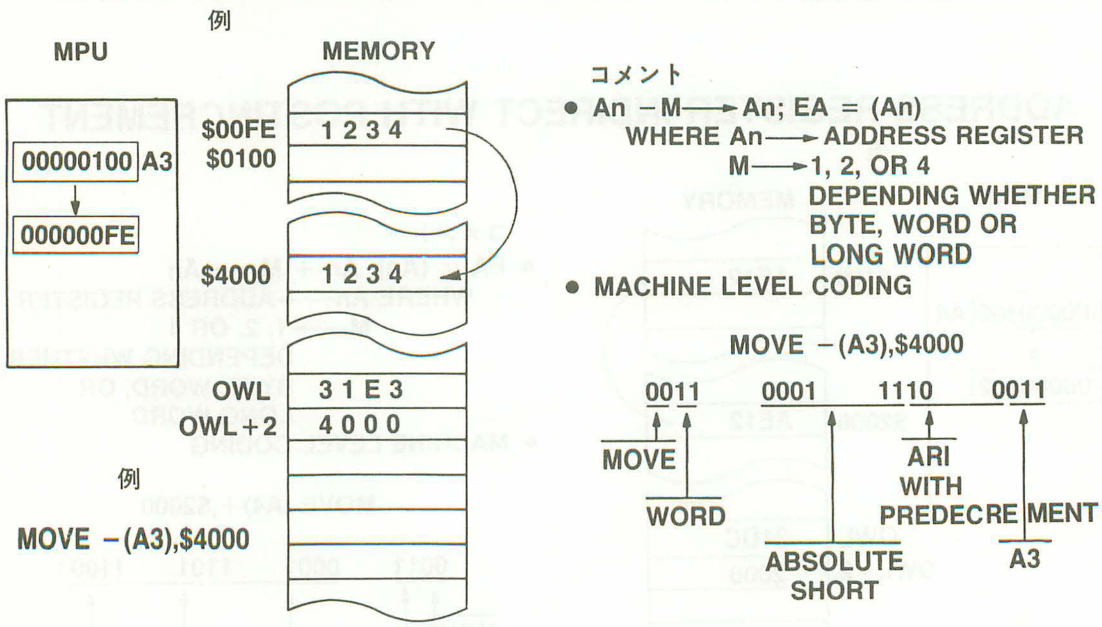
アドレス・レジスタ An

オペランドの長さ
（1，2または4）

メモリ・アドレス



ADDRESS REGISTER INDIRECT WITH PREDECREMENT



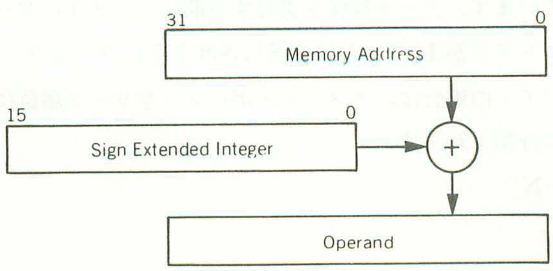
■ディスプレースメント（変位）付きアドレス・レジスタ間接　このアドレス・モードでは，操作ワードの後に1ワードの拡張が必要です．オペランド・データを示すアドレスは，アドレス・レジスタと16ビット・ディスプレースメント整数の和になります．これは JMP, JSR 命令を除き，データ・レファレンスに分類されます．

ジェネレーション	：EA＝(An)＋d
RTL 表記	：An @ (d)
モード	：101
レジスタ	：n

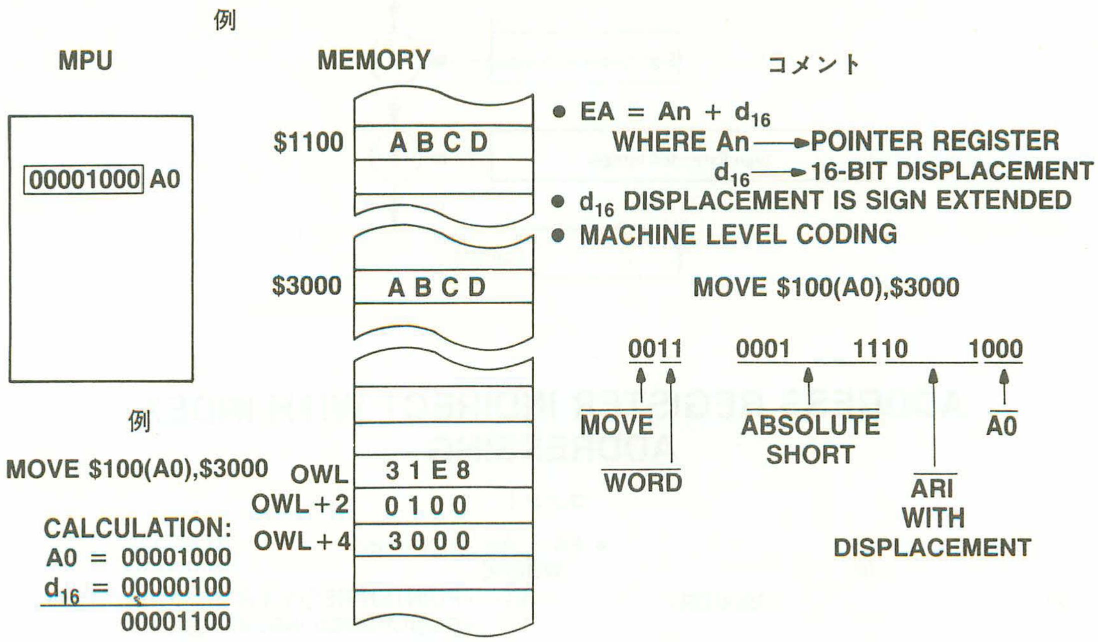
アドレス・レジスタ An

ディスプレースメント

メモリ・アドレス



ADDRESS REGISTER INDIRECT WITH DISPLACEMENT



■**インデックス付きアドレス・レジスタ間接**　このアドレッシング・モードでは，下図のようなフォーマットの1ワードの拡張ワードが必要になります．

15	14–12	11	10	9	8	7–0
D/A	Register	W/L	0	0	0	Displacement Integer

ビット15

　インデックス・レジスタの表示

　0—データ・レジスタ

　1—アドレス・レジスタ

ビット14～12

　インデックス・レジスタの番号

ビット11

　インデックス・サイズ

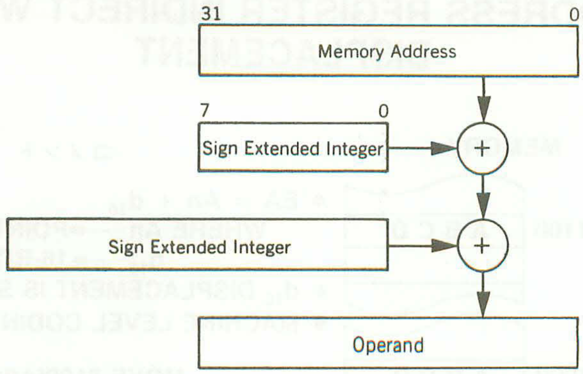
　0—インデックス・レジスタの下位整数

　1—インデックス・レジスタの上位整数

オペランド・データを示すアドレスは，アドレス・レジスタと符号拡張された8ビットのディスプレースメントとインデックス・レジスタの和になります．このレファレンスは，JSR，JMP 命令を除き，データ・レファレンスに分類されます．

ジェネレーション	：EA=(An)+(Ri)+d
RTL 表記	：An @（d，Ri．W）
	An @（d，Ri．L）
モード	：110

レジスタ　　　　　：n
アドレス・レジスタ　An

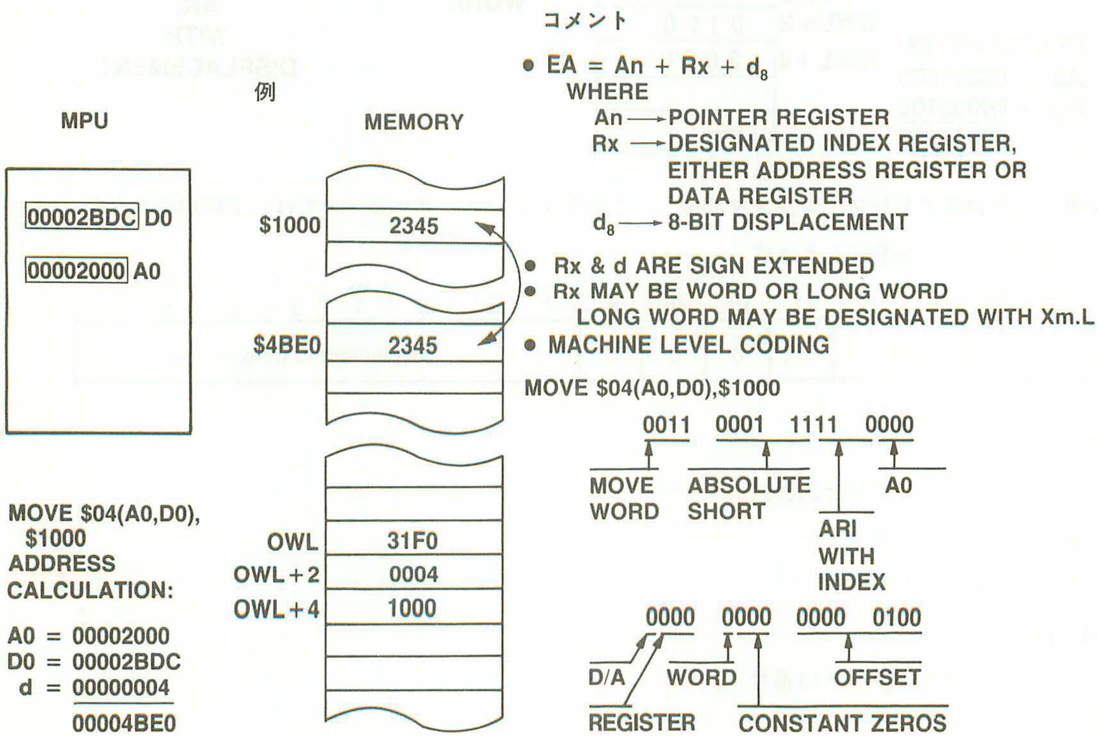


拡張ワード

インデックス・レジスタ

メモリ・アドレス

ADDRESS REGISTER INDIRECT WITH INDEX ADDRESSING



(3) スペシャル・アドレス・モード

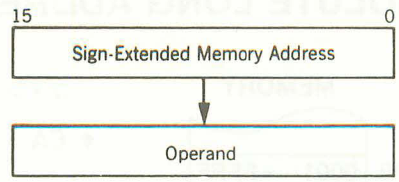
スペシャル・アドレッシング・モードは，実効アドレス・レジスタ・フィールドを使用しレジスタ番号の代わりに特定のアドレッシング・モードを指定します．

▩アブソリュート(絶対)・ショート・アドレス　このアドレッシング・モードは，1ワードの拡張ワードが必要です．オペランド・データを示すアドレスは，拡張ワードに格納されています．16ビットの拡張ワードは命令実行の前に符号により拡張されます．このレファレンスは，JMPとJSR命令を除きデータ・レファレンスに分類されます．

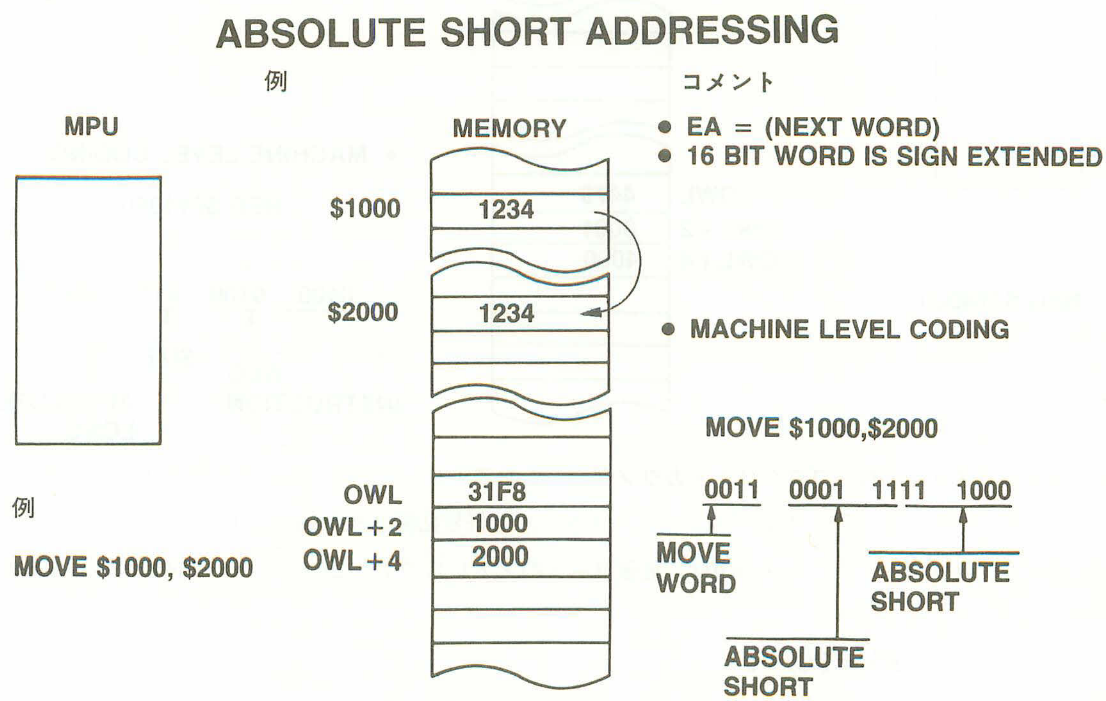
ジェネレーション　：EA　given
RTL 表記　　　　：×××．W
モード　　　　　：111

レジスタ　　　　: 000

拡張ワード



メモリ・アドレス



■アブソリュート・ロング・アドレス　　このアドレッシング・モードでは，操作ワードの後に2ワードの拡張ワードによりオペランド・データを示しアドレスを指定します．アドレスの上位部分が第1拡張ワードで，下位部分は第2拡張ワードです．このレファレンスは，JMP，JSR 命令を除きデータ・レファレンスに分類されます．

ジェネレーション　: EA　given

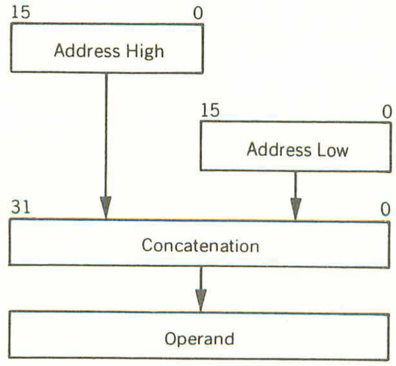
RTL 表記　　　　: ×××. L

モード　　　　　: 111

レジスタ　　　　: 001

第1拡張ワード

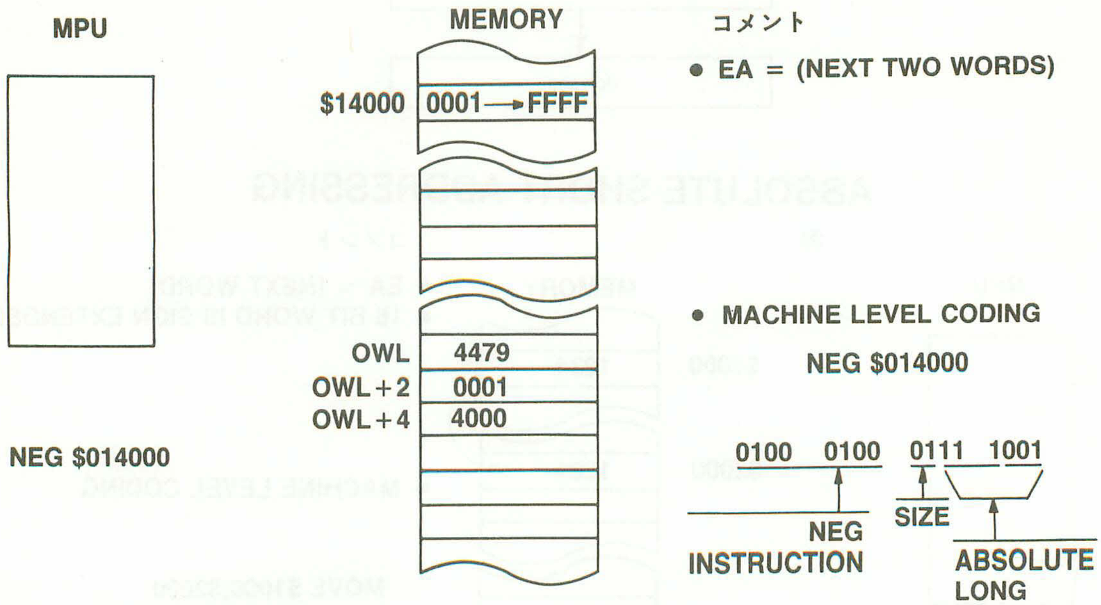
第2拡張ワード



メモリ・アドレス

M68,000 ABSOLUTE LONG ADDRESSING

例



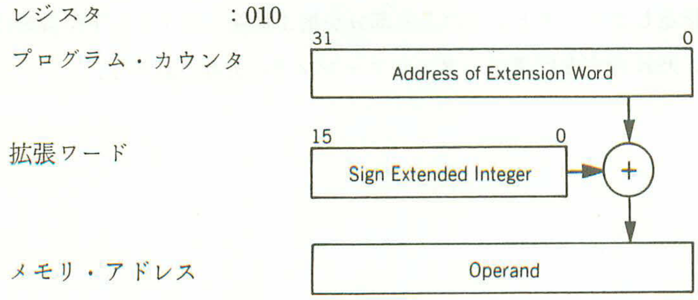
■ディスプレースメント付きプログラム・カウンタ　このアドレッシング・モードでは，1ワードの拡張ワードが必要です．オペランドのアドレスはプログラム・カウンタと符号拡張された16ビット・ディスプレースメントの和で示されます．プログラム・カウンタの値は，拡張ワードのアドレスです．このレファレンスは，プログラム・レファレンスに分類されます．

ジェネレーション　：EA＝(PC)＋d

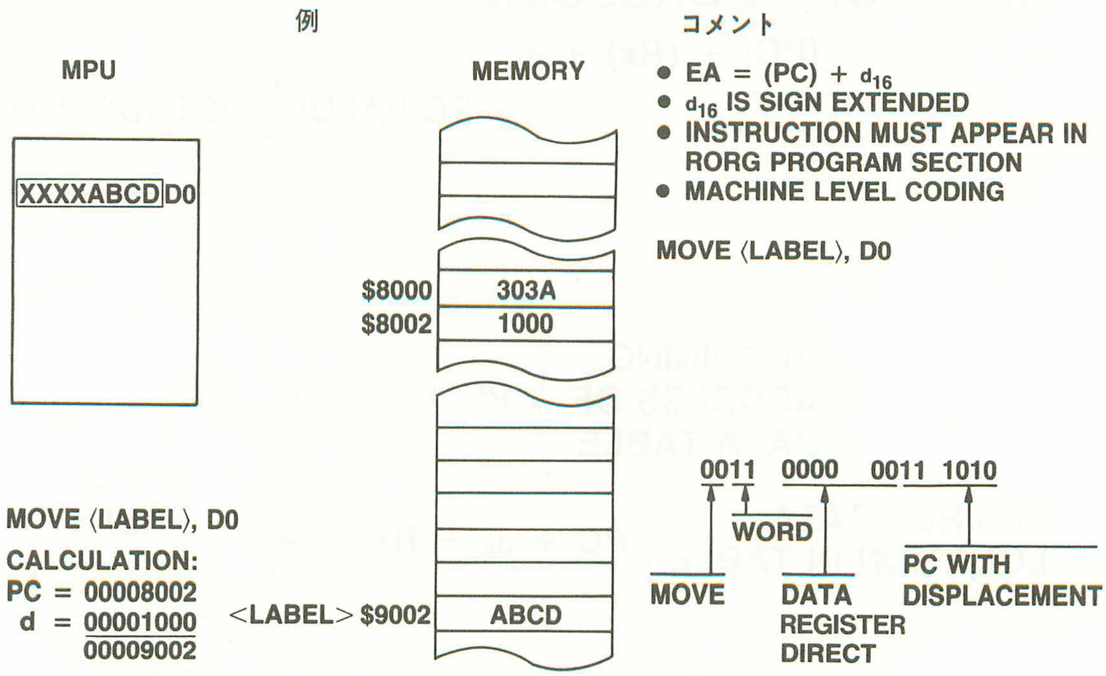
RTL 表記　：PC @ (d)

モード　：111

レジスタ　：010



PROGRAM COUNTER WITH DISPLACEMENT



■インデックス付きプログラム・カウンタ　このアドレッシング・モードでは，下図のようなフォーマットの1ワードの拡張ワードが必要です．

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	Register			W/L	0	0	0	Displacement Integer							

ビット15

　インデックス・レジスタの表示

　0—データ・レジスタ

　1—アドレス・レジスタ

ビット14〜12

　インデックス・レジスタの番号

ビット11

　インデックス・サイズ

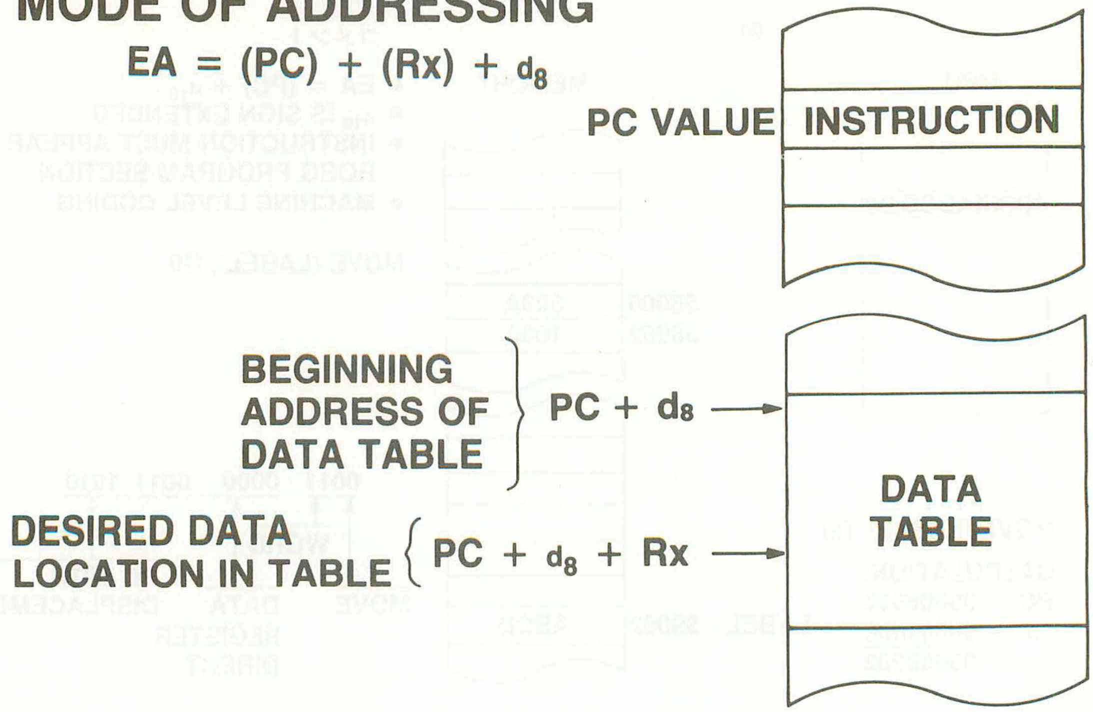
　0—インデックス・レジスタの整数下位ワード

　1—インデックス・レジスタのロング値

実効アドレスは，プログラム・カウンタと拡張された8ビット・ディスプレースメント整数とインデックス・レジスタの和になります．プログラム・カウンタの値は，拡張ワードのアドレスを示します．このレファレンスはプログラム・レファレンスに分類されます．

PC WITH INDEX MODE OF ADDRESSING

$EA = (PC) + (Rx) + d_8$



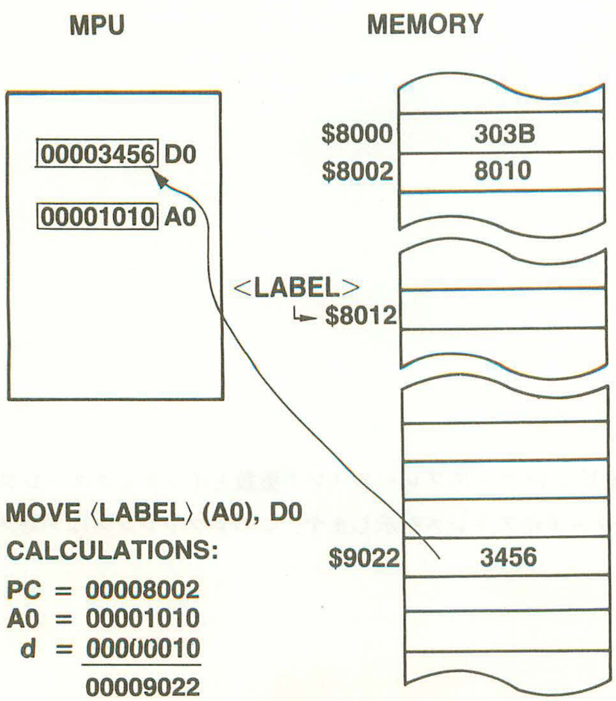
PROGRAM COUNTER WITH INDEX

コメント

- $EA = (PC) + (Rx) + d_8$
  WHERE
  PC ⟶ CURRENT PROGRAM COUNTER
  Rx ⟶ DESIGNATED INDEX REGISTER (EITHER DATA OR ADDRESS REGISTER)
  d_8 ⟶ 8-BIT DISPLACEMENT
- Rx AND d_8 ARE SIGN EXTENDED
- Rx MAY BE WORD OR LONG WORD
  LONG WORD IS DESIGNATED WITH Rx.L
- INSTRUCTION WORD MUST APPEAR IN RORG PROGRAM SECTION
- MACHINE LEVEL CODING

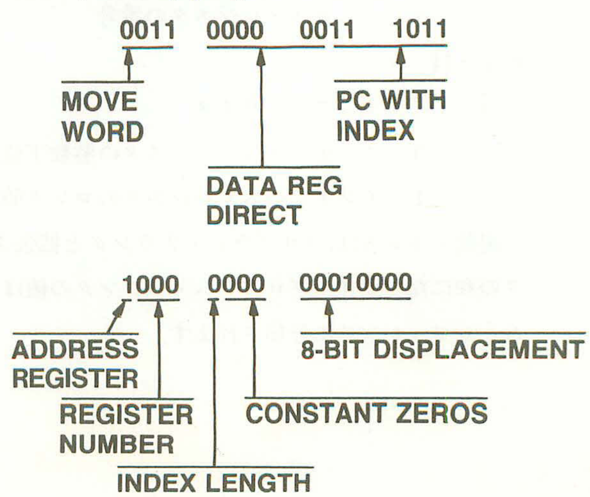
MOVE ⟨LABEL⟩ (A0), D0

例



MOVE ⟨LABEL⟩ (A0), D0
CALCULATIONS:

PC = 00008002
A0 = 00001010
d = 00000010
00009022



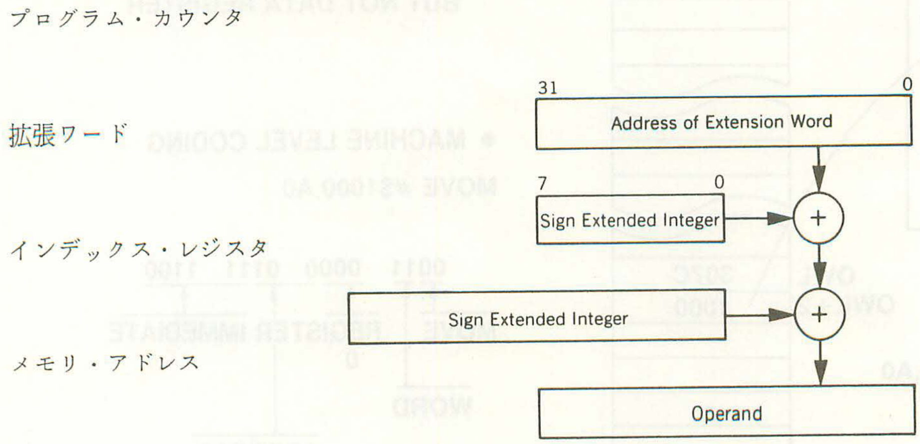
ジェネレーション　：EA＝(PC)＋(Ri)＋d

RTL 表記　：PC＠（d，Ri．W）

　　　　　　PC＠（d，Ri．L）

モード　：111

レジスタ　：011



■イミディエイト・データ　このアドレッシング・モードでは，操作されるデータのサイズに応じ，1ワードまたは2ワードの拡張ワードが必要になります．

バイト・オペレーション―オペランドは，拡張ワードの下位バイトにあります．

ワード・オペレーション―オペランドは，拡張ワードです．

ロング・ワード・オペレーション―オペランドは，2つの拡張ワードです．

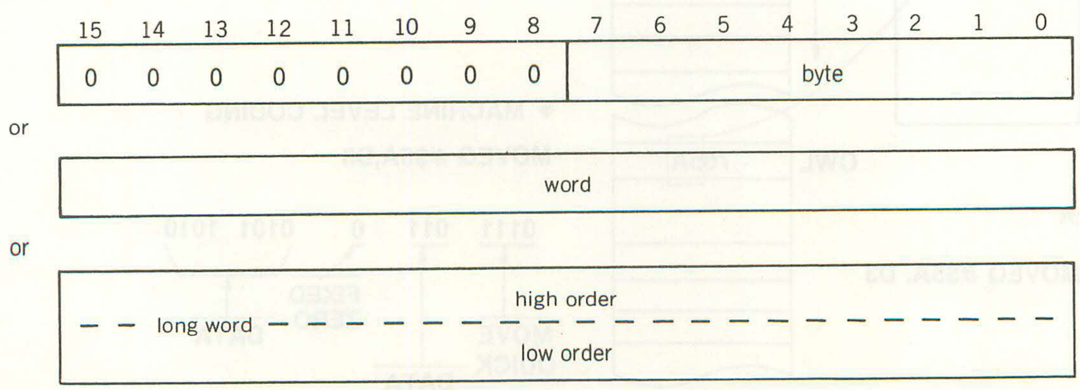
ジェネレーション　：オペランドで与えられる

RTL 表記　：＃×××

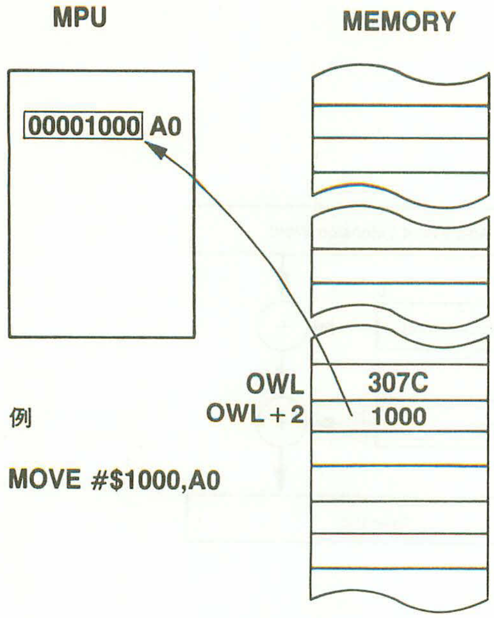
モード　：111

レジスタ　：000

拡張ワードのフォーマットを下図に示します．



MC68,000 IMMEDIATE ADDRESSING

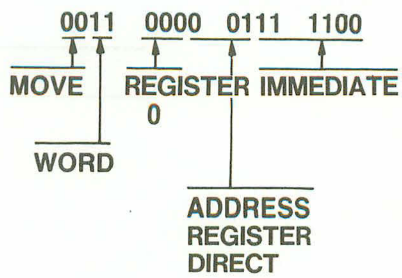


コメント

- DATA = NEXT WORD(S)
- DATA IS SIGN EXTENDED FOR ADDRESS REGISTER BUT NOT DATA REGISTER

- MACHINE LEVEL CODING

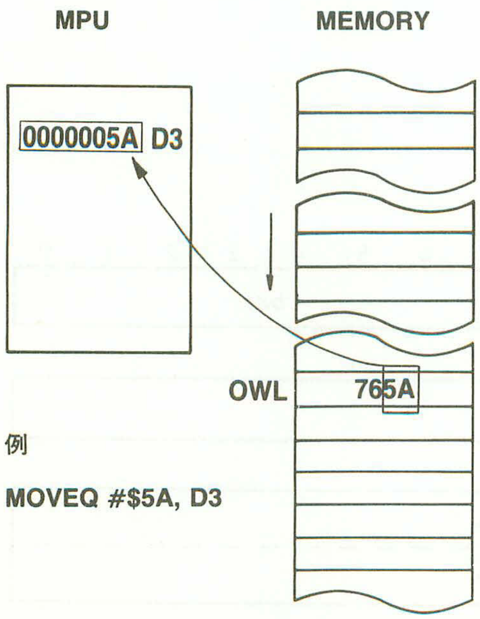
MOVE #$1000,A0



QUICK IMMEDIATE ADDRESSING

■クイック・イミディエイト

例

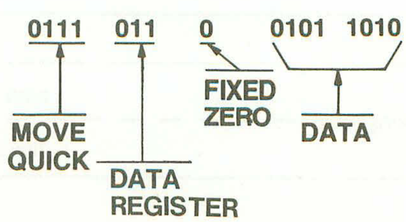


コメント

- INHERENT DATA
- DATA IS SIGN EXTENDED TO LONG WORD
- DESTINATION MUST BE A DATA REGISTER

- MACHINE LEVEL CODING

MOVEQ #$5A,D3



*INCREASING ADDRESSES

■コンディション・コードとステータス・レジスタ　MC68000の命令では，実効アドレス・フィールドでステータス・レジスタまたはコンディションコード・レジスタを指定します．

これらの命令を以下に示します．

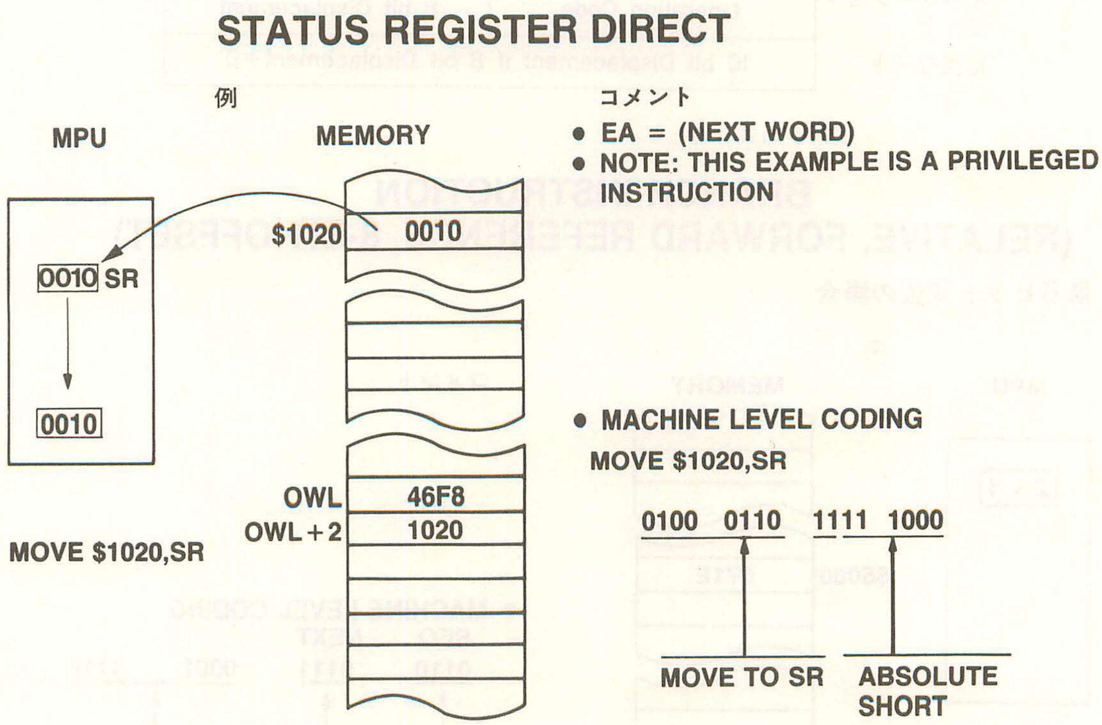
ANDI to CCR　EORI to CCR　ORI to CCR

ANDI to SR　EORI to SR　ORI to SR

RTL 表記 ：CCR あるいは SR

モード　：111

レジスタ　：100



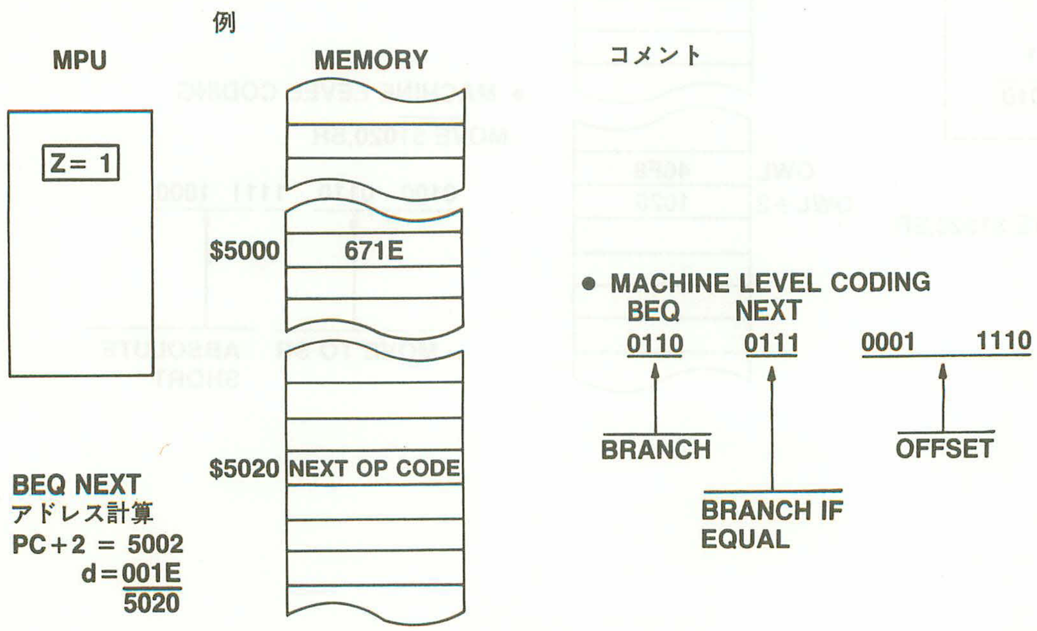
●ブランチ命令のアドレッシング
このアドレッシングはリラティブ・アドレッシングでオペレーションワード内に8ビットのディスプレースメント(2の補数表示)があります．これが0の場合は1ワードの拡張が必要です．これは16ビットのディスプレースメント(2の補数表示)になります．

分岐命令フォーマット

	15 ... 8	7 ... 0
オペレーションワード	Operation Code	8 bit Displacement
拡張ワード	16 bit Displacement if 8 bit Displacement=0	

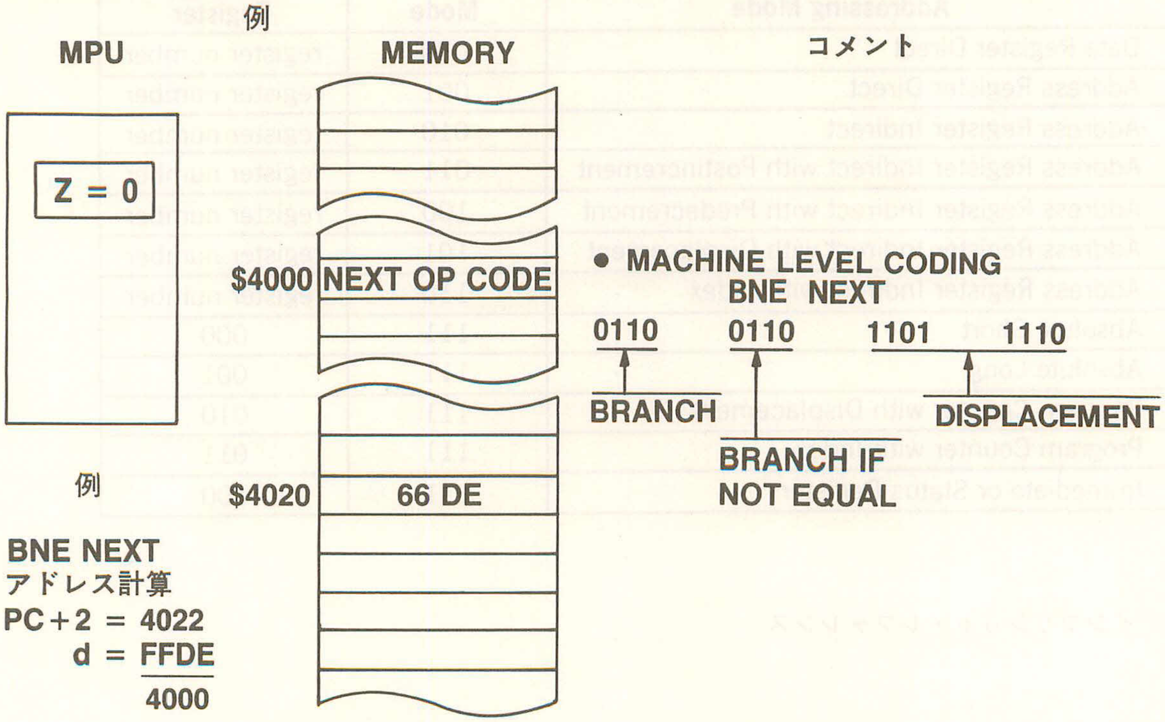
BRANCH INSTRUCTION
(RELATIVE, FORWARD REFERENCE, 8-BIT OFFSET)

■8ビット変位の場合



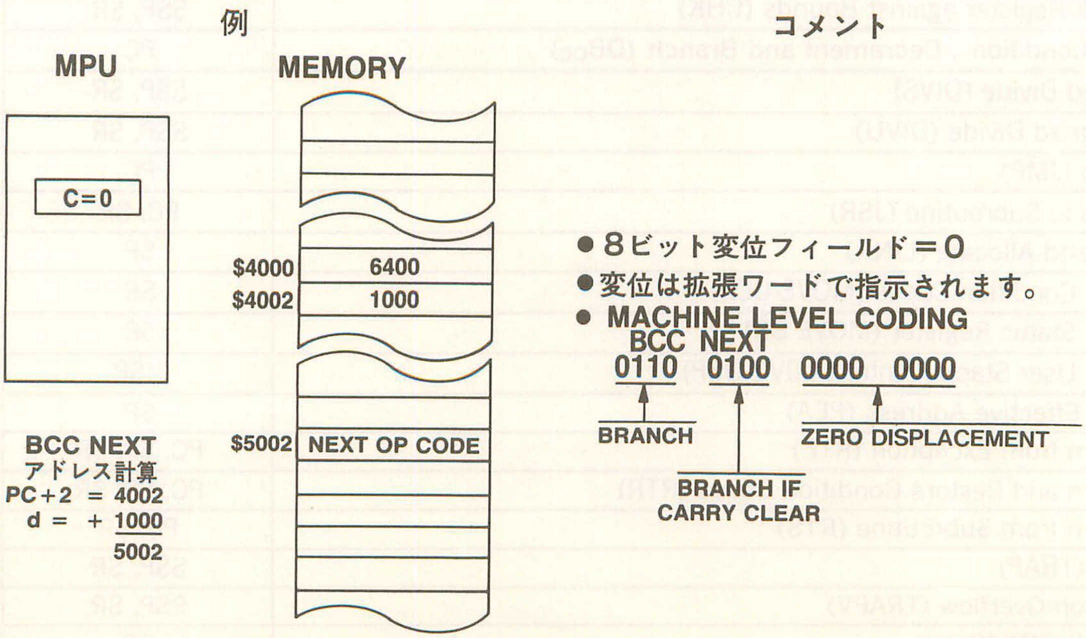
BRANCH INSTRUCTION (RELATIVE, BACKWARD REFERENCE 8-BIT OFFSET)

■8ビット変位の場合



BRANCH INSTRUCTION (RELATIVE, FORWARD REFERENCE, 16-BIT OFFSET)

■16ビット変位の場合



（4）　実効アドレスのまとめ

いままで述べた実効アドレッシング・モードを表2-1に示します．

表2-1　アドレスのアドレッシング・モードのコード別コード一覧表

Addressing Mode	Mode	Register
Data Register Direct	000	register number
Address Register Direct	001	register number
Address Register Indirect	010	register number
Address Register Indirect with Postincrement	011	register number
Address Register Indirect with Predecrement	100	register number
Address Register Indirect with Displacement	101	register number
Address Register Indirect with Index	110	register number
Absolute Short	111	000
Absolute Long	111	001
Program Counter with Displacement	111	010
Program Counter with Index	111	011
Immediate or Status Register	111	100

2.11　インプリシット・レファレンス

表 2-2　インプリシット命令一覧表

Instruction	Implied Register(s)
Branch Conditional (Bcc), Branch Always (BRA)	PC
Branch to Subroutine (BSR)	PC, SP
Check Register against Bounds (CHK)	SSP, SR
Test Condition , Decrement and Branch (DBcc)	PC
Signed Divide (DIVS)	SSP, SR
Unsigned Divide (DIVU)	SSP, SR
Jump (JMP)	PC
Jump to Subroutine (JSR)	PC, SP
Link and Allocate (LINK)	SP
Move Condition Codes (MOVE CCR)	SR
Move Status Register (MOVE SR)	SR
Move User Stack Pointer (MOVE USP)	USP
Push Effective Address (PEA)	SP
Return from Exception (RTE)	PC, SP, SR
Return and Restore Condition Codes (RTR)	PC, SP, SR
Return from Subroutine (RTS)	PC, SP
Trap (TRAP)	SSP, SR
Trap on Overflow (TRAPV)	SSP, SR
Unlink (UNLK)	SP

命令によっては，プログラム・カウンタ(PC)，システム・スタック・ポインタ(SP)，スーパバイザ・スタック・ポインタ(SSP)，ユーザ・スタック・ポインタ(USP)，ステータス・レジスタ(SR)を暗黙的に指定できる命令があります．表2-2にインプリシット命令とインプリシット・レジスタの一覧を示します．

2.12 スタックとキュー(配列データ)

インデックス・アドレッシング・モードによって配列データ構造をサポートするのに加えMC 68000では，アドレス・レジスタ間接・ポストインクリメント・アドレッシング・モードおよびプリデクリメント・アドレッシング・モードでスタックと配列データ構造をサポートします．スタックは，後入れ先出し(LIFO)の性格をもち，キューは先入れ先出し(FIFO)の性格をもちます．データをスタックやキューに加える時は，スタックやキューにプッシュ(push)され，除く時はスタックやキューからプル(pull)されます．

システム・スタックは，多数の命令により自動的に使用されます．ユーザ・スタックとキューは，アドレッシング・モードを用いて作成され操作されます．

(1) システム・スタック

アドレス・レジスタA 7は，システム・スタック・ポインタ(SP)です．システム・スタック・ポインタは，ステータス・レジスタの“S”ビットの状態に応じ，スーパバイザ・スタック・ポインタ(SSP)，あるいはユーザ・スタック・ポインタ(USP)として使用されます．“S”ビットがスーパバイザ状態を示している場合，SSPがアクティブなシステム・スタック・ポインタとなり，USPをレファレンスすることはできません．“S”ビットがユーザ状態を示している場合，USPがアクティブなシステム・スタック・ポインタとなりSSPはレファレンスできません．どちらのシステム・スタックもメモリ・アドレスの大きい方から順に小さい方へスタックされます．アドレス・モードSP@−は，アクティブ・システム・スタックにデータをプッシュし，SP@+はアクティブ・スタックからデータをプルします．

プログラム・カウンタは，サブルーチン・コールの時，アクティブ・システム・スタック・ポインタのアドレスにセーブされ，リターン時にはプログラム・カウンタは，アクティブ・システム・スタック・ポインタのアドレスよりリストアされます．一方，トラップや割込みが起こると，プログラム・カウンタとステータス・レジスタは，スーパバイザ・スタック領域へセーブされます．このようにユーザ状態とスーパバイザ状態ではそれぞれのプログラム実行は独立しています．ユーザ・プログラムでは，自由にユーザ・スタック・ポインタを使用することができます．

システム・スタック上のデータ配列を適切にしておくためには，スタックでのデータ入力ポイントは常にワード単位で操作するように制限されています．このため，バイト・データをシステム・スタックにプッシュしたり，システム・スタックからプルする場合，ワード・データの上位半分が使用され，下位半分は使用されません．

(2) ユーザ・スタック

ポストインクリメントおよびプリデクリメントのアドレス・レジスタ間接アドレッシング・モードを使用することによりユーザ・スタックを作成し操作することができます．ユーザは，アドレス・レジスタ(A 0～A 6)を，メモリの上位アドレスから下位アドレスへ順に，あるいはメモリの下位アドレスから上位アドレスへ順に進むスタック・ポインタとして利用することができます．

ユーザ・スタックを使用する場合，下記の点に注意しなければなりません．

・プリデクリメントを使用する場合，最初にデクリメントされたアドレス・レジスタの内容がスタックのポインタ

として示されます．

・ポストインクリメントを使用する場合，最初にアドレス・レジスタの内容がスタックのポインタとして示され，その後にアドレス・レジスタをインクリメントします．

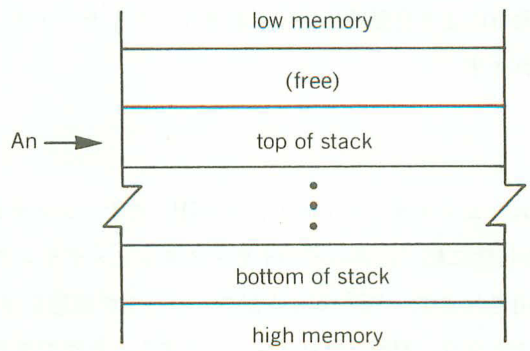
・バイト・データを使用する場合は，ワードとロング・ワードを一緒に使用することになるのでリターンするときに，配列誤りを起こさないように，バイト・データをワードの形でスタックに入れなければなりません．ワードおよびロング・ワードのアクセスは，ワード単位の偶数アドレスで行います．

メモリの上位アドレスから下位へ向かってスタック操作を行なう場合，次の処理が実施されます．

An @−　スタックにデータをプッシュします．

An @+　スタックからデータをプルします．

プッシュまたはプルのスタック操作の後，アドレス・レジスタ An は，先頭または最後尾スタックのアドレスを示します．これを下図に示します．

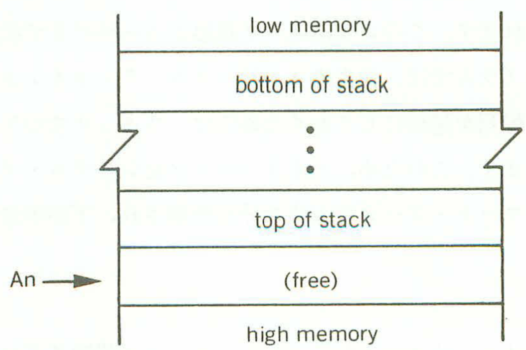


メモリの下位アドレスから上位へ向かってスタック操作を行なう場合，次の処理が行われます．

An @+　スタックへデータをプッシュします．

An @−　スタックからデータをプルします．

プッシュまたはプルのスタック操作の後，アドレス・レジスタ An は，次に使用するスタックのエリアを示します．



(3) キュー

ポストインクリメントまたはプリデクリメント・アドレス・レジスタ間接のアドレッシング・モードを利用することにより，ユーザのキューを作成し操作することができます．ユーザは2個のアドレス・レジスタ（A 0～A 6のうち2個）をメモリの上位アドレスから下位アドレスへ順に，あるいはメモリの下位アドレスから上位アドレスへ順に

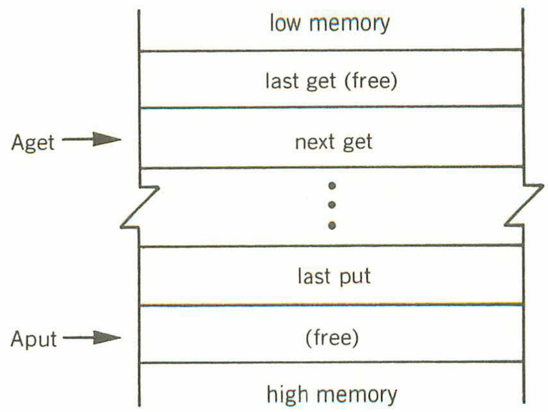
進む待ち行列機能として利用することができます．キューはある一端から取り入れを，もう一端から取り出しを行いますが，このために2個のアドレス・レジスタを使用します．終端管理のアドレス・レジスタをプット（put）・ポインタと呼び，先端管理のアドレス・レジスタをゲット（get）・ポインタと呼びます．

メモリの下位アドレスから上位へ向かって待ち行列操作を行なう場合，次の処理が実施されます．

A put @+　キューへデータをプットします．

A get @+　キューからデータをゲットします．

プット操作の後，プット・アドレス・レジスタは，キューの次につながるデータの位置を示し，ゲット操作の後，ゲット・アドレス・レジスタは，キューから次にゲットするデータの位置を示します．この操作を下図に示します．



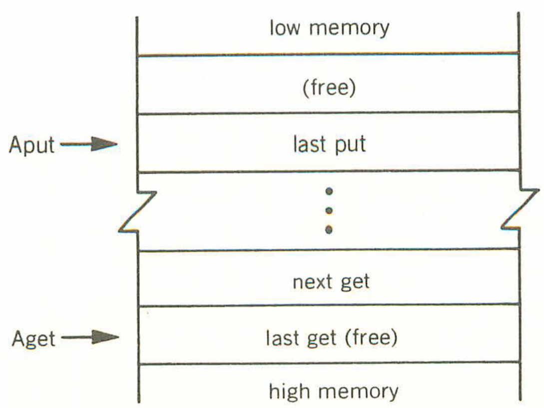
もし，キューを循環バッファとして使用するときは，プットまたはゲット操作を実行する前にアドレス・レジスタからバッファの長さ（バイト単位）を差し引いて調整する必要がある場合があります．

メモリの上位アドレスから下位へ向かってキュー操作を行なう場合，次の処理が実施されます．

A put @−　キューへデータをプットします．

A get @−　キューからデータをゲットします．

プット操作後のプット・アドレス・レジスタは，キューにつながれている最後のデータ位置を示し，ゲット操作後のゲット・アドレス・レジスタはキューから最後にゲットされたデータ位置を示します．この操作を下図に示します．



また，キューを循環バッファとして使用する場合，ゲットまたはプット操作を最初に実行し，もし必要となればその後バッファの長さ（バイト単位）をアドレス・レジスタに加算することによって修正します．

第3章　命令セットの概要

本章では，MC 68000 命令セット構造の概要について説明します．命令は以下のオペレーションの項目を実行するためにオペレーションごとに命令集団を形成しています．

- Data Movement
- Integer Arithmetic
- Logical
- Shift and Rotate
- Bit Manipulation
- Binary Coded Decimal
- Program Control
- System Control

第2章で述べた，フレキシブルなアドレッシング・モードと組み合わせることにより，命令機能が完全なものとなり，プログラムを開発する上で非常にフレキシブルな基盤が与えられます．

3.1　データ転送オペレーション

MOVE 命令によって基本的なデータ操作（転送および格納）ができます．MOVE 命令と実効アドレッシング・

表3-1　データ転送オペレーション

Instruction	Operand Size	Operation
EXG	32	Rx → Ry
LEA	32	EA → An
LINK	—	An → SP@- SP → An SP + d → SP
MOVE	8, 16, 32	$(EA)_s$ → EA_d
MOVEM	16, 32	(EA) → An, Dn An, Dn → EA
MOVEP	16, 32	(EA) → Dn Dn → EA
MOVEQ	8	#xxx → Dn
PEA	32	EA → SP@-
SWAP	32	Dn [31:16] ↔ Dn [15:0]
UNLK	—	An → SP SP @+ → An

（注）　s＝ソース，d＝ディスティネーミョン，〔　〕＝ビット番号

モードによってアドレスおよびデータ処理が可能になります．データ MOVE 命令により，バイト，ワード，ロング・ワード・オペランド・データをメモリからメモリ，メモリからレジスタ，レジスタからメモリ，レジスタからレジスタへ転送できます．アドレス MOVE 命令では，ワードとロング・ワード・オペランド・データの転送ができます．一般的な MOVE 命令に加え，次のような特殊なデータ MOVE 命令があります．

MOVEM (move multiple registers), MOVEP (move peripheral data), EXG (exchange registers), LEA (load effective address), PEA(push effective address), LINK (link stack), UNLK (unlink stack), MOVEQ (move quick).

表 3-1 に，データ転送オペレーションの概要を示します．

3.2　整数演算オペレーション

整数演算オペレーションには，4つの基本操作，ADD（加算：add），SUB（減算：subtract），MUL（乗算：multiply），DIV（除算：divide）に加えて，CMP（演算比較：arithmetic compare），CLR（クリア：clear），NEG（ネゲート：negate）が含まれます．加算命令と減算命令は，アドレス・オペレーションおよびデータ・オペレーシ

表 3-2　整数演算オペレーション

Instruction	Operand Size	Operation
ADD	8, 16, 32 16, 32	Dn + (EA) → Dn (EA) + Dn → EA (EA) + #xxx → EA An + (EA) → An
ADDX	8, 16, 32 16, 32	Dx + Dy + X → Dx Ax@- Ay@- + X → Ax@
CLR	8, 16, 32	0 → EA
CMP	8, 16, 32 16, 32	Dn - (EA) (EA) - #xxx Ax@+ - Ay@+ An - (EA)
DIVS	32 ÷ 16	Dn/(EA) → Dn
DIVU	32 ÷ 16	Dn/(EA) → Dn
EXT	8 → 16 16 → 32	$(Dn)_8 \rightarrow Dn_{16}$ $(Dn)_{16} \rightarrow Dn_{32}$
MULS	16 * 16 → 32	Dn * (EA) → Dn
MULU	16 * 16 → 32	Dn * (EA) → Dn
NEG	8, 16, 32	0 - (EA) → EA
NEGX	8, 16, 32	0 - (EA) - X → EA
SUB	8, 16, 32 16, 32	Dn - (EA) → Dn (EA) - Dn → EA (EA) - #xxx → EA An - (EA) → An
SUBX	8, 16, 32	Dx - Dy - X → Dx Ax@- - Ay@- - X → Ax@
TAS	8	(EA) - 0, 1 → EA[7]
TST	8, 16, 32	(EA) - 0

（注）〔　〕＝ビット番号

ョンの両方に使用できます．データ・オペレーションはすべてのオペランド・データ・サイズでの操作が可能です．アドレス・オペレーションは，アドレス・サイズのオペランド（16または32ビット）に限定されます．データ比較，アドレス比較およびメモリ比較・オペレーションも同様に使用できます．クリアとネゲート命令はすべてのデータ・オペランド・サイズで使用できます．乗算・除算命令は符号付きと符号なしオペランド・データに使用できます．つまりロング・ワード積を生じるワード・サイズの乗算，およびワード・サイズの商とワード・サイズの余りを生じるロング・ワード・サイズの被除数とワード・サイズの除数による除算を処理できます．

拡張命令セットを使用することによって精度やサイズが異なっていても算術演算を実行することができます．これらの命令は ADDX (add extended), SUBX (subtract extended), EXT (sign extend), NEGX (negate binary with extend) です．

オペランドを0と比較しその結果に従ってコンディション・コードをセットする TST (test operand) 命令も備わっています．TAS (test and set) 命令はマルチ・プロセッサ・システムにおいて必要な同期命令です．表3-2に整数演算オペレーションの概要を示します．

3.3 論理演算オペレーション

論理演算操作命令（AND，OR，EOR，NOT）は，すべてのサイズの整数データ・オペランドに対して使用できます．イミディエイト命令セット（ANDI，ORI，EORI）もすべてのサイズのイミディエイト・データの論理演算操作を可能にします．表3-3に論理演算操作の概要を示します．

表3-3 論理演算オペレーション

Instruction	Operand Size	Operation
AND	8, 16, 32	$Dn \wedge (EA) \to Dn$ $(EA) \wedge Dn \to EA$ $(EA) \wedge \#xxx \to EA$
OR	8, 16, 32	$Dn \vee (EA) \to Dn$ $(EA) \vee Dn \to EA$ $(EA) \vee \#xxx \to EA$
EOR	8, 16, 32	$(EA) \oplus Dy \to EA$ $(EA) \oplus \#xxx \to EA$
NOT	8, 16, 32	$\sim(EA) \to EA$

Note:
～ = invert

3.4 シフトおよびローテイト・オペレーション

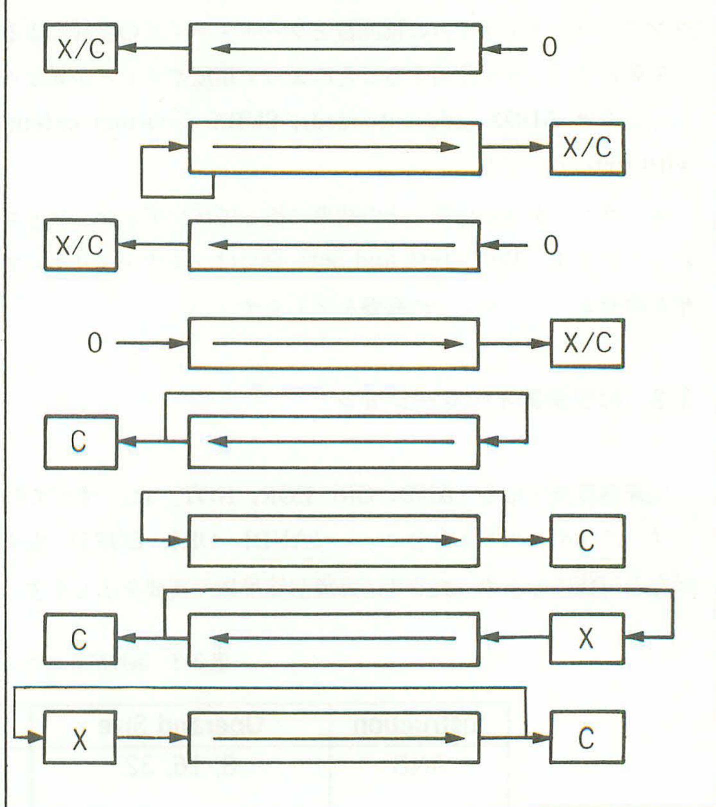
双方向へのシフト・オペレーションは，演算命令 ASR と ASL，論理シフト命令 LSR と LSL によって実行できます．ローテイト命令（拡張あり，および拡張なし）としては ROXR，ROXL，ROR，ROL が使用できます．すべてのシフトとローテイト・オペレーションはレジスタとメモリのいずれにおいても実行できます．レジスタ・シフトとレジスタ・ローテイトは，すべてのサイズのオペランドを取ることができ，シフト・カウントは0～8ビットまで命令によって指定するか，あるいは0～63ビットまでデータ・レジスタで指定できます．

メモリ・シフトとメモリ・ローテイトは，ワード・オペランドのみ操作でき1ビットだけシフトまたはローテイトできます．

表3-4にシフトおよびローテイト・オペレーションの概要を示します．

表3-4 シフトとローテイト・オペレーション

Instruction	Operand Size	Operation
ASL	8, 16, 32	
ASR	8, 16, 32	
LSL	8, 16, 32	
LSR	8, 16, 32	
ROL	8, 16, 32	
ROR	8, 16, 32	
ROXL	8, 16, 32	
ROXR	8, 16, 32	



3.5 ビット処理オペレーション

次の命令を使用して，ビット処理オペレーションを実行します．BTST (bit test), BSET (bit test and set), BCLR (bit test and clear), BCHG(bit test and change) です．表3-5にビット処理オペレーションの概要を示します．

表3-5 ビット処理オペレーション

Instruction	Operand Size	Operation
BTST	8, 32	~ bit of (EA) → Z
BSET	8, 32	~ bit of (EA) → Z 1 → bit of EA
BCLR	8, 32	~ bit of (EA) → Z 0 → bit of EA
BCHG	8, 32	~ bit of (EA) → Z ~ bit of (EA) → bit of EA

3.6 BCD（2進化10進）オペレーション

BCD での多重精度算術演算は次の命令を使用して実行します．ABCD (add decimal with extend), SBCD (subtract decimal with extend), NBCD (negate decimal with extend) です．表3-6に BCD オペレーションの概要を示します．

表3-6 BCD オペレーション

Instruction	Operand Size	Operation
ABCD	8	$Dx_{10} + Dy_{10} + X \rightarrow Dx$ $Ax@-_{10} + Ay@-_{10} + X \rightarrow Ax@$
SBCD	8	$Dx_{10} - Dy_{10} - X \rightarrow Dx$ $Ax@-_{10} - Ay@-_{10} - X \rightarrow Ax@$
NBCD	8	$0 - (EA)_{10} - X \rightarrow EA$

3.7 プログラム・コントロール・オペレーション

プログラム・コントロール・オペレーションは，一連の条件つきブランチ命令および無条件ブランチ命令とリターン命令を使用し実行します．これらの命令については表3-7で説明します．

条件つき命令では，以下の条件に対してセットおよびブランチを行います．

CC — carry clear
CS — carry set
EQ — equal
F — never true
GE — greater or equal
GT — greater than
HI — high
LE — less or equal
LS — low or same
LT — less than
MI — minus
NE — not equal
PL — plus
T — always true
VC — no overflow
VS — overflow

表3-7　プログラム・コントロール・オペレーション

Instruction	Operation
Conditional	
Bcc	Branch conditionally (14 conditions) 8- and 16-bit displacement
DBcc	Test condition, decrement and branch. 16-bit displacement
Scc	Set byte conditionally (16 conditions)
Unconditional	
BRA	Branch always 8- and 16-bit displacement
BSR	Branch to subroutine 8- and 16-bit displacement
JMP	Jump
JSR	Jump to subroutine
Returns	
RTR	Return and restore condition codes
RTS	Return from subroutine

3.8　システム・コントロール・オペレーション

システム・コントロール・オペレーションは，特権命令，トラップ命令および，ステータス・レジスタを使用するか修正する命令を用いて実行します．これらの命令については表3-8に示します．

表3-8　システム・コントロール・オペレーション

Instruction	Operation
Privileged	
RESET	Reset external devices
RTE	Return from exception
STOP	Stop program execution
ORI to SR	Logical OR to status register
MOVE USP	Move user stack pointer
ANDI to SR	Logical AND to status register
EORI to SR	Logical EOR to status register
MOVE EA to SR	Load new status register
Trap Generating	
TRAP	Trap
TRAPV	Trap on overflow
CHK	Check register against bounds
Status Register	
ANDI to CCR	Logical AND to condition codes
EORI to CCR	Logical EOR to condition codes
MOVE EA to CCR	Load new condition codes
ORI to CCR	Logical OR to condition codes
MOVE SR to EA	Store status register

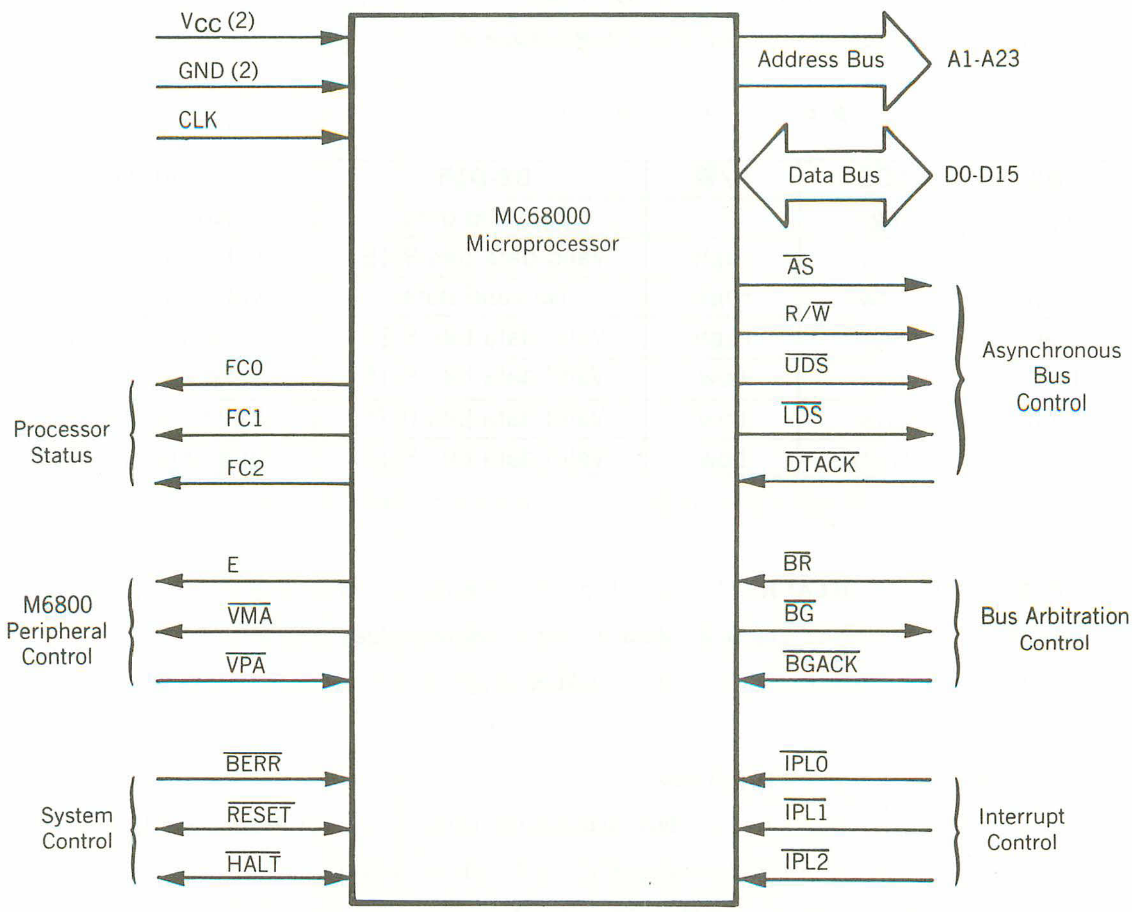
第4章　信号およびバス・オペレーションの説明

本章では，入／出力信号についての概要，ならびにマシン・サイクル，マシン操作におけるバス動作について説明します．

4.1　信号の説明

入／出力信号は，図 4-1 に示すような機能により分類することができます．以下各信号についてその概要を説明します．なお，詳しくは他の章も参照してください．

図 4-1　入／出力信号



（1）　アドレス・バス（A1～A23）

この23ビット単方向性３ステート・バスは，16Mバイト（８Mワード）のデータを直接にアドレスできます．このバスは，割込みサイクルを除いたすべてのサイクルにおいて，バス・オペレーション用のアドレスを決定します．割込みサイクルは，アドレス・ラインＡ１，Ａ２，Ａ３によって，どのレベルの割込みが実行されているかを示しています．その間，他のアドレス・ラインＡ４～A23はすべて“H”レベルにセットされています．

（2）　データ・バス（D0～D15）

この16ビット双方向性3ステート・バスは，汎用データ・バスです．このバスにより，ワード長またはバイト長のデータの転送（送／受信）ができます．割込みアクノリッジ・サイクル中は，外部デバイスからデータ・ラインD0～D7にベクタ番号が送り出されます．

（3）　非同期バス・コントロール

非同期データの転送（送／受信）は，アドレス・ストローブ，リード／ライト，上位データ・ストローブ，下位データ・ストローブおよびデータ転送アクノリッジのコントロール信号を用います．

▒アドレス・ストローブ（$\overline{AS}$）　この信号は，アドレス・バス上に有効なアドレスが存在していることを示します．

▒リード／ライト（R/$\overline{W}$）　この信号はデータ・バスの転送方向がリード・サイクルかライト・サイクルかを規定します．R/$\overline{W}$ 信号は，次に説明する$\overline{UDS}$／$\overline{LDS}$とともに動作します．

▒上位／下位データ・ストローブ（$\overline{UDS}$，$\overline{LDS}$）　これらの信号は，表4-1で示しているようにデータ・バス上のデータをコントロールします．R/$\overline{W}$ ラインが“H”の場合，プロセッサはデータ・バスよりデータを読み込み，“L”の場合にはプロセッサはデータ・バスにデータを書き込みます．

表4-1　データ・バスのデータ・ストローブ・コントロール

$\overline{UDS}$	$\overline{LDS}$	R/$\overline{W}$	D8-D15	D0-D7
High	High	—	No valid data	No valid data
Low	Low	High	Valid data bits 8-15	Valid data bits 0-7
High	Low	High	No valid data	Valid data bits 0-7
Low	High	High	Valid data bits 8-15	No valid data
Low	Low	Low	Valid data bits 8-15	Valid data bits 0-7
High	Low	Low	Valid data bits 0-7*	Valid data bits 0-7
Low	High	Low	Valid data bits 8-15	Valid data bits 8-15*

＊　これらの条件は，現在の実行結果であり，将来のデバイスには発生しない場合もあります．

▒データ転送アクノリッジ（$\overline{DTACK}$）　この入力は，データ転送（送／受信）が完了していることを示します．プロセッサがリード・サイクル中に$\overline{DTACK}$（data transfer acknowledge）を認めると，データをラッチしバス・サイクルを終了させます．ライト・サイクル中に$\overline{DTACK}$を認めるとバス・サイクルを終了させます．

（4）　バス・アービトレーション・コントロール

バス・アービトレーション・コントロール（bus arbitration control）は，以下の3つの信号によって行われ，どのデバイスがバス・マスタ・デバイスになるかを決定します．（4.2(2)項参照）

▒バス・リクエスト（$\overline{BR}$）　この入力信号は，すべてのバス・マスタとなりうるデバイスとワイヤードORされて使用されます．この入力信号は，他のデバイスがバス・マスタになることを要求していることをプロセッサへ示します．

▒バス・グラント（$\overline{BG}$）　この出力信号は，他のバス・マスタとなることのできるすべてのデバイスに対し，プロセッサが現在のバス・サイクルの終了した時点でバスのコントロールを解除することを示します．

▒バス・グラント・アクノリッジ（$\overline{BGACK}$）　この入力信号は，他のデバイスがバス・マスタとなったことを

示します．ただし，以下の4つの条件が満足されない限りこの信号は有効とはなりません．

(1) バス・グラントが受信されていること

(2) アドレス・ストローブが休止状態（inactive）であること．つまりプロセッサがバスを使用していないこと．

(3) $\overline{\mathrm{DTACK}}$ が休止状態になっていること．つまりメモリや周辺デバイスがバスを使用していないこと．

(4) $\overline{\mathrm{BGACK}}$ が休止状態であること．つまり他のデバイスがバス・マスタとなることを要求していないこと．

（5） 割込みコントロール（$\overline{\mathrm{IPL}0}$，$\overline{\mathrm{IPL}1}$，$\overline{\mathrm{IPL}2}$）

これらの入力信号は，割込みを要求するデバイスの優先レベルを示します．レベル7が最も優先度が高く，レベル0は割込み要求がないことを示しています．最下位ビットは $\overline{\mathrm{IPL}0}$ で，最上位ビットは $\overline{\mathrm{IPL}2}$ で示されます．
割込み操作についての詳細は第5章で述べます．

（6） システム・コントロール

システム・コントロール信号は，プロセッサをリセット（reset）またはホルト（halt）させたり，プロセッサにバス・エラーの発生を知らせます．システム・コントロール信号には次の3つがあります．

▒バス・エラー（$\overline{\mathrm{BERR}}$）　この入力信号は，実行中のバス・サイクルに問題が生じたことをプロセッサに知らせます．ここでいう問題とは次に示すようなことです．

(1) デバイスが応答しない．

(2) 割込みベクトル番号を取り込みできない．

(3) メモリ・マネジメント・ユニット（MMU）により違法アクセス要求が検出された．

(4) 他のアプリケーションからのエラー．

バス・エラー信号とホルト信号は相互に作用してエクセプション・プロセッシング（例外処理）を実行し，エラー処理すべきかエラーを起こしたバス・サイクルをリラン（再実行）すべきかを決定します（バス・エラーとホルトの相互の関係動作については，4.2(3)項を参照）．

▒リセット（$\overline{\mathrm{RESET}}$）　この双方向信号ラインは，外部からリセット信号を入力してやることによりプロセッサをリセット（システム・イニシャライズ・シーケンスを開始する）することができます．RESET 命令によってプロセッサで作成されたリセット信号は，この端子から出力され他のすべての外部デバイスをリセットします．しかし，プロセッサの内部状態は変化しません．ホルトとリセット信号を同時に加えることによりシステム全体（プロセッサおよびシステムに接続されているデバイス）がリセットされます（リセット操作については，4.2(4)項参照）．

▒ホルト（$\overline{\mathrm{HALT}}$）　外部デバイスからホルト信号（双方向性ライン）を駆動すると，プロセッサは動作中のバス・サイクルが完了した時点で停止します．プロセッサがホルトした場合，すべてのコントロール信号は休止状態になり，3ステート・ラインはハイ・インピーダンス状態となります．（ホルト信号とバス・エラー信号との相互の働きについては4.2(3)項参照）．

プロセッサがダブル・バス・フォルト状態になり命令実行を停止した場合，プロセッサはホルト・ラインより動作状態を出力し，外部のデバイスに対してプロセッサが停止したことを知らせます．

（7） M6800周辺デバイスのコントロール

これらのコントロール信号を使用することにより非同期式の MC 68000 が同期式の M 6800 周辺デバイスを使用することができます．これらのコントロール信号には以下のものがあります．

▒イネーブル（E）　この信号はすべての M 6800 周辺デバイスが使用する同期用クロック信号（enable）です．

この信号の1サイクルはMC 68000の10クロックに相当します（6クロックが“L”，4クロックが“H”）.

■バリッド周辺アドレス（$\overline{\text{VPA}}$）　この入力信号によりプロセッサがアクセスするデバイスおよび領域がM 6800ファミリ・デバイスであり，データの転送（送／受信）がイネーブル（E）信号と同期して行われていることを示します．また，この入力信号は，プロセッサが割込み処理にあたって自動ベクタ動作を行うことを指示するためにも使用されます（第6章参照）.

■バリッド・メモリ・アドレス（$\overline{\text{VMA}}$）　この出力信号は，M 6800周辺デバイスに対してアドレス・バスに有効アドレスがあることを示し，プロセッサがイネーブルと同期して動作していることを示します．この信号は，周辺デバイスがM 6800ファミリ・デバイスであることを示す$\overline{\text{VPA}}$入力にのみ応答します.

（8）　プロセッサのステータス信号（FC 0，FC 1，FC 2）

これらのファンクション・コード出力信号は，表4-2に示すように実行中のモード（ユーザまたはスーパバイザ）とバス・サイクルを表示します．ファンクション・コード信号で示される情報は，アドレス・ストローブ（$\overline{\text{AS}}$）がアクティブであれば常に有効です.

表4-2　ファンクション・コード出力

FC2	FC1	FC0	Cycle Type
Low	Low	Low	(Undefined, Reserved)
Low	Low	High	User Data
Low	High	Low	User Program
Low	High	High	(Undefined, Reserved)
High	Low	Low	(Undefined, Reserved)
High	Low	High	Supervisor Data
High	High	Low	Supervisor Program
High	High	High	Interrupt Acknowledge

（9）　クロック入力（CLK）

クロック入力はTTLとコンパチブルな信号で，内部でバッファされ，プロセッサが必要とする内部クロックを作成します．クロック入力は一定周波数の信号でなければなりません.

（10）　信号のまとめ

表4-3に今までの項目において説明した全部の信号をまとめます.

4.2　バス・オペレーション

以下の各項においてリード／ライト・サイクル，リード-モディファイ-ライト（read-modify-write）・サイクル，バス・アービトレーション，バス・エラーとホルト状態およびリセット・オペレーションのそれぞれにおけるコントロール信号とバス・オペレーションについて説明します.

（1）　データ転送オペレーション

デバイス間のデータ転送（送／受信）には以下の端子を使用します.

表4-3　入／出力信号一覧表

Signal Name	Mnemonic	Input/Output	Active State	Three State
Address Bus	A1-A23	output	high	yes
Data Bus	D0-D15	input/output	high	yes
Address Strobe	$\overline{AS}$	output	low	yes
Read/Write	R/$\overline{W}$	output	read-high write-low	yes
Upper and Lower Data Strobes	$\overline{UDS}$, $\overline{LDS}$	output	low	yes
Data Transfer Acknowledge	$\overline{DTACK}$	input	low	no
Bus Request	$\overline{BR}$	input	low	no
Bus Grant	$\overline{BG}$	output	low	no
Bus Grant Acknowledge	$\overline{BGACK}$	input	low	no
Interrupt Priority Level	$\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$	input	low	no
Bus Error	$\overline{BERR}$	input	low	no
Reset	$\overline{RESET}$	input/output	low	no*
Halt	$\overline{HALT}$	input/output	low	no*
Enable	E	output	high	no
Valid Memory Address	$\overline{VMA}$	output	low	yes
Valid Peripheral Address	$\overline{VPA}$	input	low	no
Function Code Output	FC0, FC1, FC2	output	high	yes
Clock	CLK	input	high	no
Power Input	V_{CC}	Input	—	—
Ground	GND	Input	—	—

*open drain

- アドレス・バス　A 1～A23
- データ・バス　D 0～D 15
- コントロール信号

アドレス・バスとデータ・バスは，それぞれ独立した並列バスであり非同期動作でデータを転送（送／受信）します．どのようなバス・サイクルでも，バス・マスタはすべての信号動作をそのバス・サイクル内で完了させなくてはなりません．

これからリード／ライト・サイクルおよびリード-モディファイ-ライト・サイクルについて説明します．1つのバス・サイクルになっているリード-モディファイ-ライト・サイクルは，MC 68000によってインターロックされたマルチプロセッサ間のコミュニケーションのために使われる手法です．

＜注　意＞

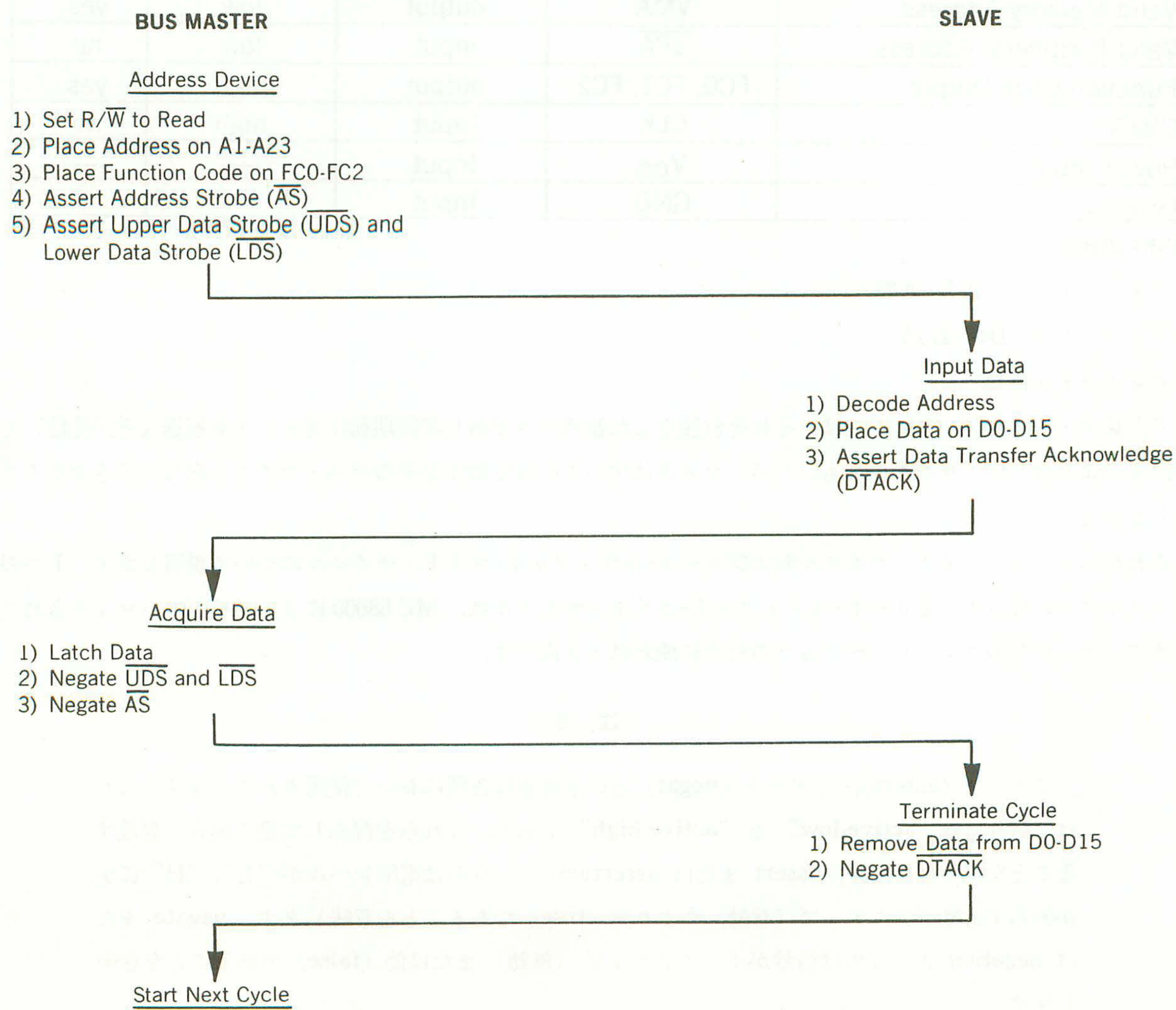
アサート（assert），ネゲート（negat）という言葉は各所において使用されています．これは，信号には“active-low”と“active-high”があり，これらを混在して扱う場合，混乱することを防ぐためです．assert または assertion というのは電圧レベルが“L”，“H”にかかわらず信号がアクティブ（有効）つまり真（true）であることを意味します．negate または negation というのは信号がインアクティブ（無効）または偽（false）であることを意味します．

■リード・サイクル リード・サイクルでプロセッサは，メモリまたは周辺デバイスからデータを受け取ります．プロセッサは常に複数バイトのデータを読み取ります．もし命令がワード・データ操作を指定している場合，プロセッサは2バイトを読み取ります．バイト・データ操作が命令により指定されている場合，プロセッサは内部のA0ビットを使用し，どのバイトを読み取るかを決定しそのバイトに必要なデータ・ストローブを出力します．バイト・データ操作で，A0ビットが0の場合には上位データ・ストローブ（$\overline{\mathrm{UDS}}$）を出力し，A0ビットが1の場合には下位データ・ストローブ（$\overline{\mathrm{LDS}}$）を出力します．データを受けるとプロセッサは，指示された場所にデータを格納します．

ワード・リード・サイクルのフローチャートを図4-2に，バイト・リード・サイクルのフローチャートを図4-3に示します．図4-4には，リードとライト・サイクル・タイミング，そして図4-5にはワードとバイト・リード・サイクル・タイミングの詳細を示します．

リード・サイクルで，ステート0（S0）の場合に，アドレス・バス（A1～A23）およびファンクション・コード（FC0～FC2）は，ハイ・インピーダンス状態です．リード／ライト信号（R/$\overline{\mathrm{W}}$）は“H”となり，リード・サイクルを示します．半クロック後，ステート1（S1）においてアドレス・バスおよびファンクション・コードは，ハイ・インピーダンスから解除されます．ファンクション・コードは，このサイクルがどのアドレス空間で動作しているかを示します．

図4-2 ワード・リード・サイクルのタイミング図



ステート2（S 2）において，アドレス・ストローブ（$\overline{AS}$）がアサートされ，アドレス・バス上に有効アドレスがあることを示します．さらに必要に応じて上位データ・ストローブ（$\overline{UDS}$）および下位データ・ストローブ（$\overline{LDS}$）がアサートとなります．メモリまたは周辺デバイスは，アドレス・バスとアドレス・ストローブをデコードし自分が選択されたかどうかを判断します．選択されたデバイスは，リード／ライト信号とデータ・ストローブを使用してデータ・バス上にデータを送り出すと同時にデータ転送アクノリッジ（$\overline{DTACK}$）をアサートします．ステート3および4の間で新しい信号が発生することはありません．

データ転送アクノリッジ（$\overline{DTACK}$）は，ステート5（S 5）の開始には信号がアサート状態になっていなければなりません．データ転送アクノリッジ（$\overline{DTACK}$）がアサート状態になっていないと，ステート5とステート6は待ち状態になります．ステート5では応答したデータ転送アクノリッジ（$\overline{DTACK}$）と同期がとられます．ステート6（S 6）では，次に続くプロセッサの内部バス・サイクルのための要求が，バス・インターフェース回路から出力されます．ステート6（S 6）の終了時点（ステート7の開始時点）でプロセッサ内部のデータ・バス・ホールド・レジスタに入力したデータをラッチします．

ステート7（S 7）の間に，アドレス・ストローブ（$\overline{AS}$），上位データ・ストローブ（$\overline{UDS}$）および（または）下位データ・ストローブ（$\overline{LDS}$）はネゲートとなります．アドレス・バスは，スタティック・メモリ・オペレーショ

図4-3 バイト・リード・サイクルのタイミング図

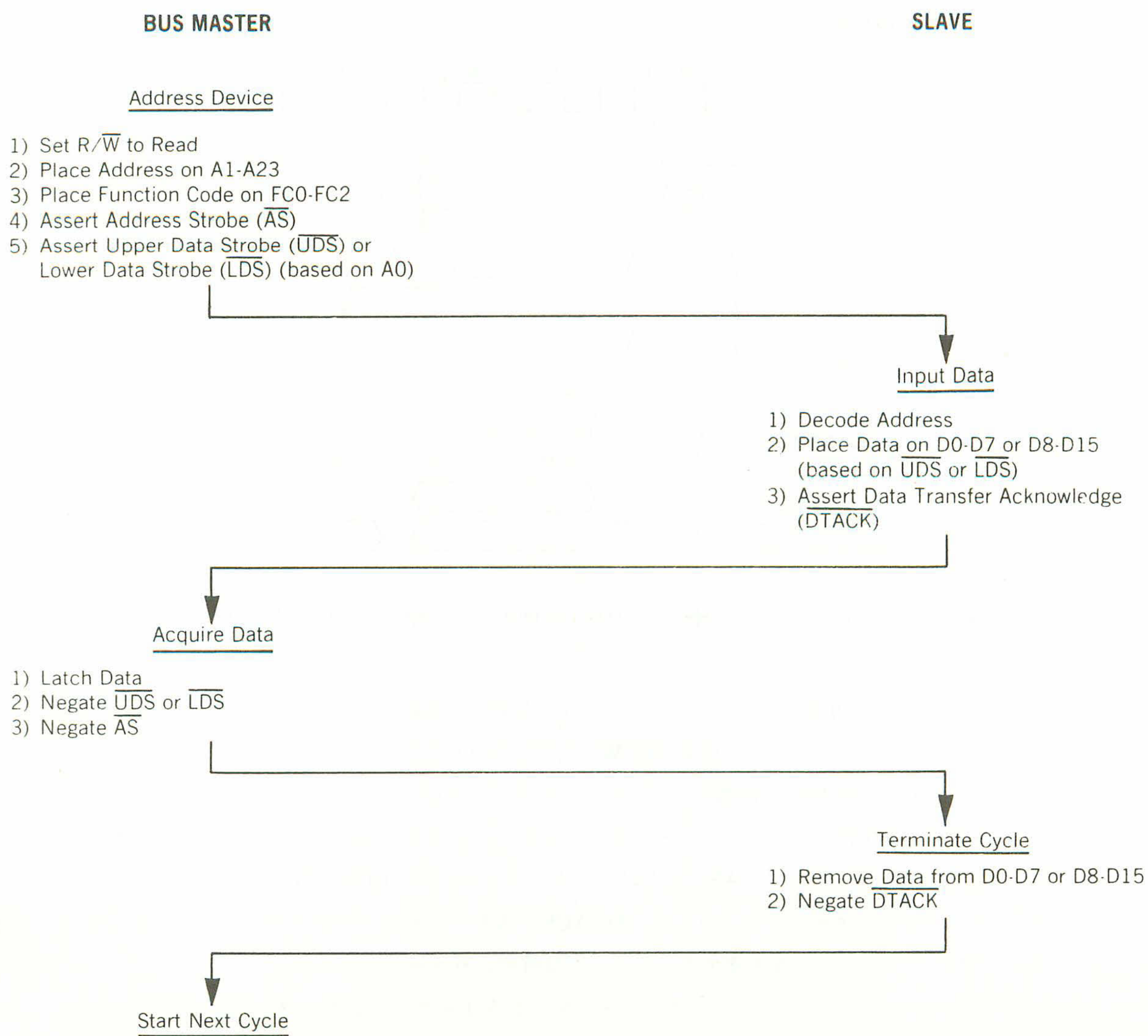


図4-4 リード・サイクルとライト・サイクルのタイミング図

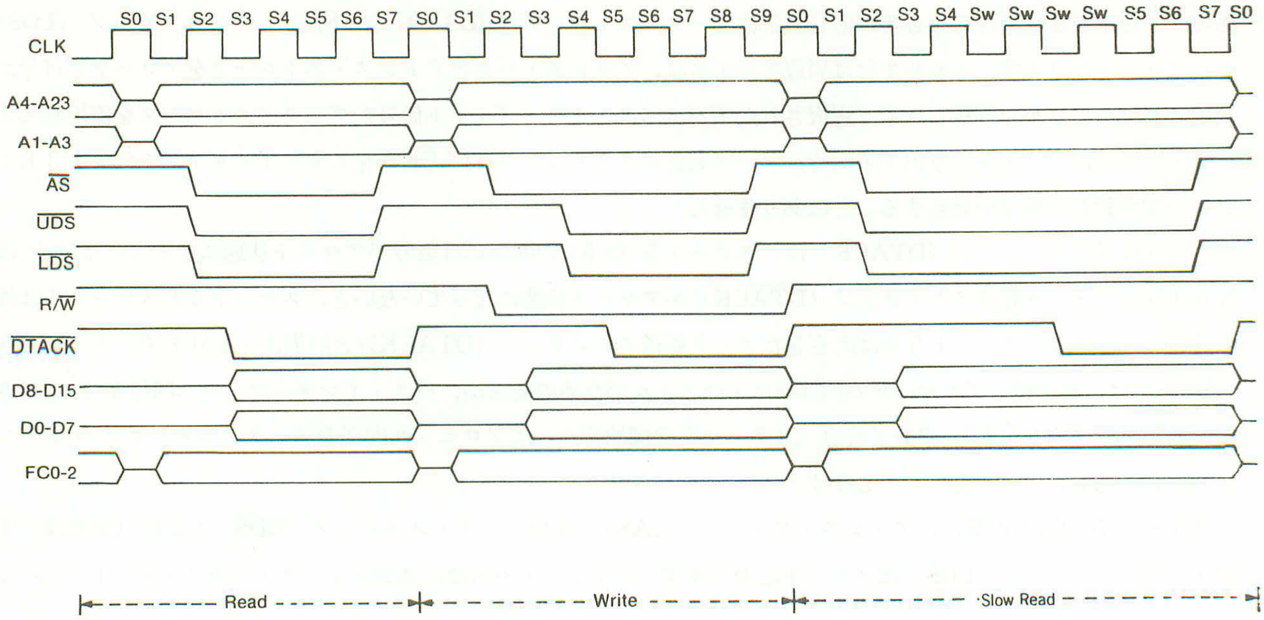
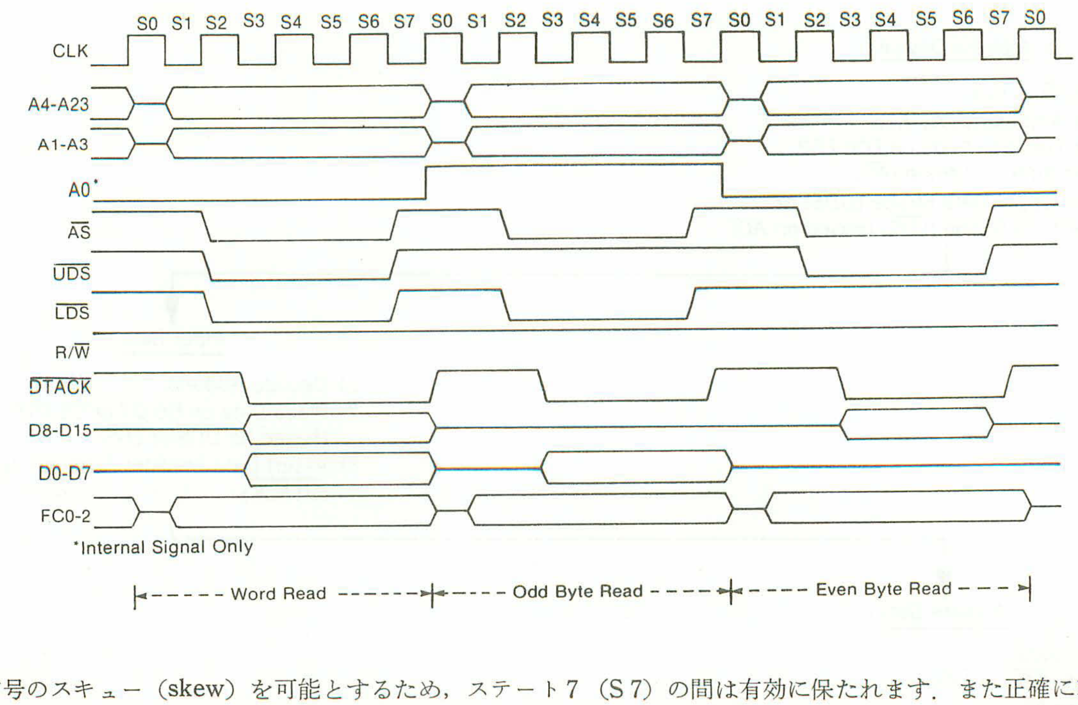


図4-5 ワード，バイト・サイズ・リード・サイクルのタイミング図



ンと信号のスキュー（skew）を可能とするため，ステート7（S 7）の間は有効に保たれます．また正確に転送オペレーションができるように，リード／ライト信号（R/$\overline{\text{W}}$）とファンクション・コード（FC 0～FC 2）もステート7（S 7）では，有効のまま保持されます．従属デバイスのリード・データは，アドレス・ストローブ（$\overline{\text{AS}}$）または上位データ・ストローブおよび下位データ・ストローブがネゲートとなるまでアサート状態に維持しています．スレーブ・デバイスは，アドレス・ストローブ（$\overline{\text{AS}}$）またはデータ・ストローブ（$\overline{\text{LDS}}$，$\overline{\text{UDS}}$）のネゲートを確認の後，1クロック以内にデータとデータ転送アクノリッジ（$\overline{\text{DTACK}}$）を取り除かなくてはなりません．取り除かなければ次のステート0（S 0），1（S 1），になるまでデータ・バスは解放されません．

アドレス・ストローブがネゲートされると，スレーブ・デバイスは解放されます．スレーブ・デバイスはリード–

モディファイ-ライト・サイクルが正確に機能するようにアドレス・ストローブ（$\overline{AS}$）がアクティブである限り選択されたままでなければなりません．（リード・モディファイ-ライト・サイクルの項に説明してあります.）

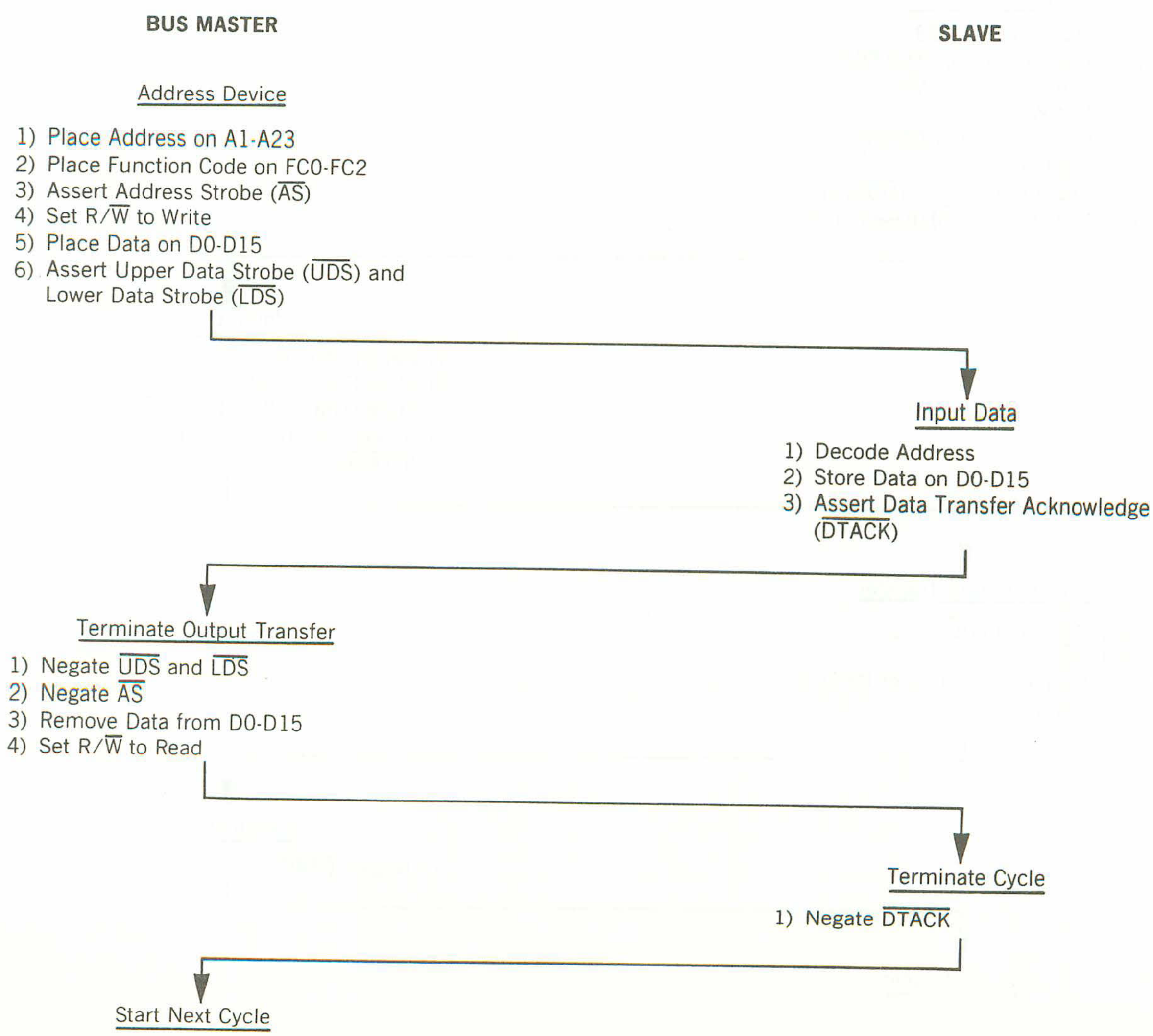
▩ライト（書き込み）・サイクル　ライト・サイクルの間でプロセッサは，メモリまたは周辺デバイスにデータを送り出します．常にプロセッサは複数バイトのデータを書き込みます．命令がワード・データ操作を指定した場合，プロセッサは2バイトを書き込みます．バイト・データ操作の場合，プロセッサは内部のA0ビットを使用してどのバイトに書き込むかを決め，そのバイトに必要なデータ・ストローブを出力します．A0ビットが0の場合，上位データ・ストローブ（$\overline{UDS}$）を出力し，A0ビットが1の場合下位データ・ストローブ（$\overline{LDS}$）を出力します．ワード・ライト・サイクルのフローチャートを図4-6に，バイト・ライト・サイクルのフローチャートを図4-7に示します．図4-4にリードとライト・サイクル・タイミングを，図4-8にワードとバイトのライト・サイクル・オペレーションをそれぞれ示します．

ライト・サイクルでステート0（S0）の時，アドレス・バス（A1～A23）とファンクション・コード（FC0～FC2）は，ハイ・インピーダンス状態です．

＜注　意＞

リード／ライト信号（R/$\overline{W}$）は，バスと前のリード・サイクルとの衝突を避けるためにス

図4-6　ワード・ライト・サイクルのフローチャート



テート2（S 2）になるまで“H”に維持されます．データ・バスはステート3（S 3）になるまで駆動されません．

半クロック後，ステート1（S 1）においてアドレス・バスとファンクション・コード（FC 0～FC 2）がハイ・インピーダンス状態から解除されます．ファンクション・コード（FC 0～FC 2）は，このバス・サイクルがどのアドレス空間で動作しているかを示します．

ステート2（S 2）では，アドレス・ストローブ（$\overline{\text{AS}}$）はアサートとなり，アドレス・バス上のアドレスが有効であることを示します．メモリまたは周辺デバイスは，自身が選ばれているかどうかをアドレス・バスとアドレス・ストローブ（$\overline{\text{AS}}$）をデコードして判断します．ステート2（S 2）の間に，リード／ライト（R/$\overline{\text{W}}$）信号は“L”になりライト・サイクルであることを示します．データ・バス・バッファが必要な場合，リード／ライト・ラインによってバッファの方向制御を行うことができます．この間，データはアサートされていないのでデータ・バッファに対し十分なターン・アラウンド・タイムをとることが可能です．ステート3（S 3）の間にデータはデータ・バス上にアサートされます．

図4-7　バイト・サイクルのフローチャート

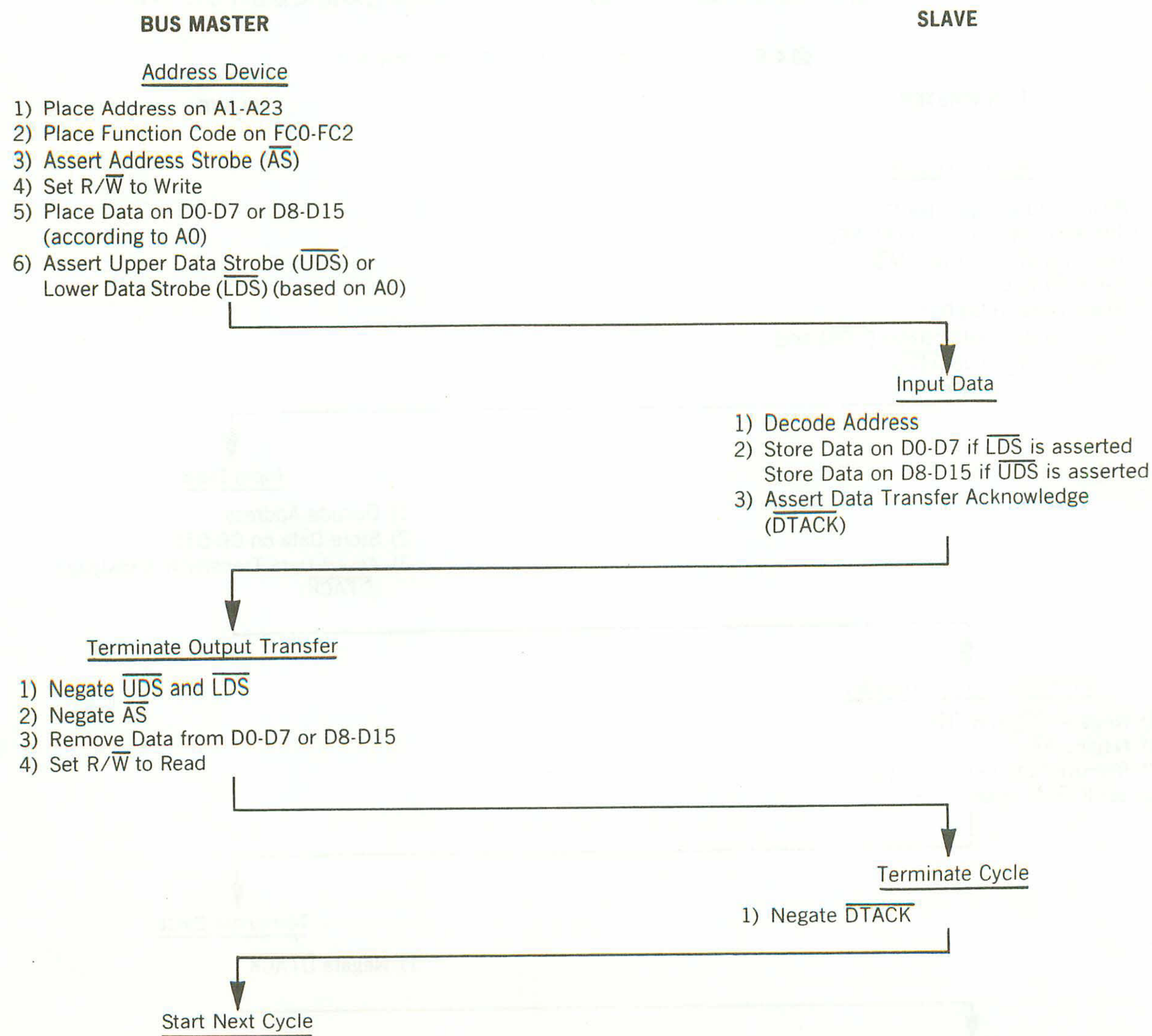
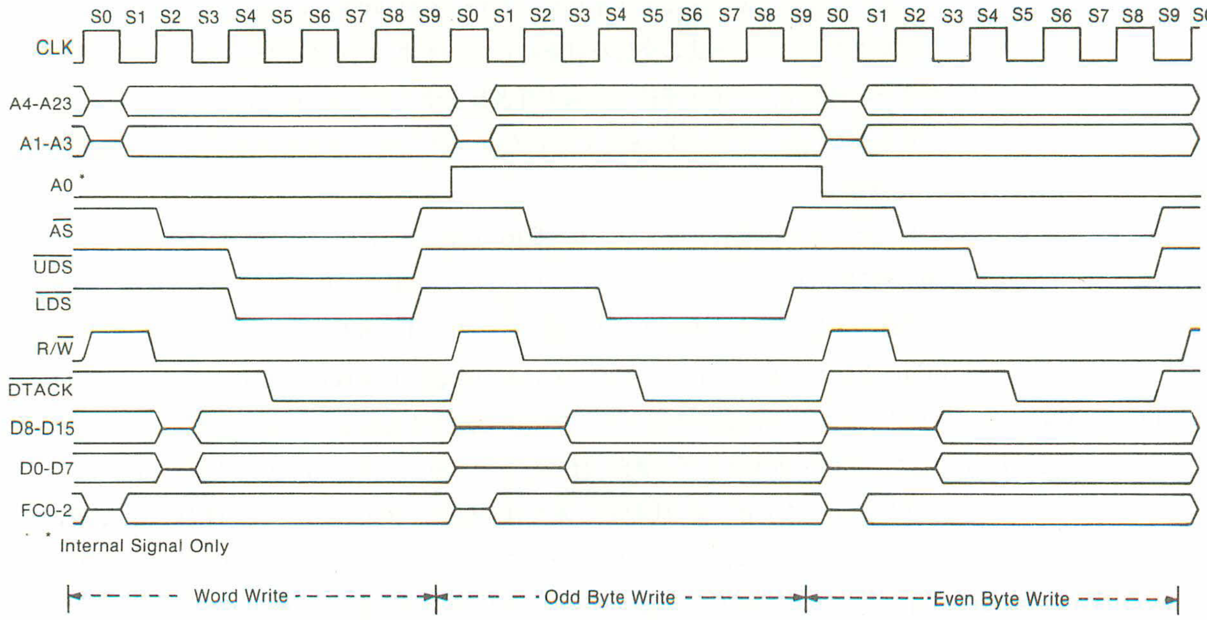


図4-8 ワードおよびバイト・ライト・サイクルのタイミング図



ステート4（S 4）では，データ・ストローブはアサートされデータ・バスが定まったことを示します．スレーブ・デバイスは，リード／ライト信号とデータ・ストローブを使用し，データ・バスからデータを取り込みます．スレーブ・デバイスは，データをストアしたときに，データ転送アクノリッジ（$\overline{\mathrm{DTACK}}$）をアサートします．ステート5（S 5），6（S 6）の間に新しいコントロール信号が発生することはありません．

データ転送アクノリッジ（$\overline{\mathrm{DTACK}}$）は，ステート7（S 7）の開始時点にはアサート状態になっていなければなりません．データ転送アクノリッジ（$\overline{\mathrm{DTACK}}$）がアサート状態になっていないと，プロセッサはデータ転送アクノリッジがアサートになるまで待ち（wait）状態に入ります．ステータス7では返答してきたデータ転送アクノリッジ（$\overline{\mathrm{DTACK}}$）の同期がとられます．これによりステート8（S 8）では，プロセッサ内部サイクルに対する要求がバス・インターフェース回路によって出されます．

ステート9の間にアドレス・ストローブ（$\overline{\mathrm{AS}}$）と上位データ・ストローブ（$\overline{\mathrm{UDS}}$）ならびに下位データ・ストローブ（$\overline{\mathrm{LDS}}$）がネゲートとなります．アドレス・バスとデータ・バスは，ステート9（S 9）の間，スタティック・メモリ・オペレーションと信号のスキュー（skew）ができるように保持（FC 0～FC 2）されています．リード／ライト（R/$\overline{\mathrm{W}}$）信号とファンクション・コード（FC 0～FC 2）も同様にステート9（S 9）の間，正しく転送オペレーションができるように保持されます．スレーブ・デバイスは，アドレス・ストローブ（$\overline{\mathrm{AS}}$）かまたは上位データ・ストローブ（$\overline{\mathrm{UDS}}$）下位データ・ストローブ（$\overline{\mathrm{LDS}}$）のネゲートを検出するまでデータ転送アクノリッジ（$\overline{\mathrm{DTACK}}$）をアサートに保持します．スレーブ・デバイスは，アドレス・ストローブ（$\overline{\mathrm{AS}}$）またはデータ・ストローブ（$\overline{\mathrm{UDS}}$），（$\overline{\mathrm{LDS}}$）がネゲートとなったことを確認した後，1クロック以内にデータ転送アクノリッジ（$\overline{\mathrm{DTACK}}$）の出力をネゲートしなくてはなりません．

注意：プロセッサはステート9（S 9）の終了時点でデータ・バスを解放しますが，データ転送アクノリッジ（$\overline{\mathrm{DTACK}}$）はステート0（S 0）もしくは1（S 1）まで解除されない可能性があります．アドレス・ストローブ（$\overline{\mathrm{AS}}$）がネゲートとなった時点でスレーブ・デバイスは解放されます．

■リード-モディファイ-ライト・サイクル　リード-モディファイ-ライト・サイクルはデータを読み出し内部の ALU（arithmetic logic unit）によって修正し，そして読み出した時と同じアドレスにデータを書き込みます．MC

68000のリード-モディファイ-ライト・サイクル全体を通じてアドレス・ストローブがアサートとなっているのでリード・サイクルとライト・サイクルに分割することはできません．TAS (test and set) 命令は，このサイクルを利用しマルチプロセッサ・システムでのプロセッサ間の交信を可能にしています．この命令は，リード-モディファイ-ライト・サイクルを使用する唯一の命令で，バイト単位でのみ実行されます．リード-モディファイ-ライト・サイクルはバイト・オペレーションとなっています．リード-モディファイ-ライト・サイクルのフローチャートを図4-9に，またタイミングを図4-10に示します．

リード-モディファイ-ライト・サイクルの場合，ステート0（S 0）の時，アドレス・バス（A1～A23）とファンクション・コード出力（FC 0～FC 2）は，ハイ・インピーダンスとなっています．リード／ライト信号（R/W）はリード・サイクルを示すため“H”となります．半クロックの後，ステート1（S 1）において，アドレス・バスとファンクション・コード（FC 0～FC 2）は，ハイ・インピーダンスの状態から解放されます．ファンクション・コード（FC0～FC 2)は，このバス・サイクルがどのアドレス・レンジで動作するかを示します．

ステート2（S 2）においてアドレス・ストローブ（$\overline{\text{AS}}$）はアサートされ，アドレス・バス上に有効アドレスのあることを示します．そして上位データ・ストローブ（$\overline{\text{UDS}}$），または下位データ・ストローブ（$\overline{\text{LDS}}$）が必要に応じてアサートされます．スレーブ・デバイスは，アドレス・バスとアドレス・ストローブ（$\overline{\text{AS}}$）をデコードして選択されたかどうかを判断します．スレーブ・デバイスは，リード／ライト信号（R/$\overline{\text{W}}$）とデータ・ストローブ（$\overline{\text{UDS}}$, $\overline{\text{LDS}}$）を使用し，データ・バス上に情報を送り出します．スレーブ・デバイスは，データ・バス上にデータを送り出すと同時にデータ転送アクノリッジ（$\overline{\text{DTACK}}$）をアサートします．ステート3（S 3）と4（S 4）では，新しい信号は出力されません．

データ転送アクノリッジ（$\overline{\text{DTACK}}$）は，ステート5（S 5）の開始時点までにアサート状態になっていなければなりません．アサート状態になっていないとプロセッサはデータ転送アクノリッジ（$\overline{\text{DTACK}}$）がアサートされるまで待ち状態となります．ステート5（S 5）では，応答したデータ転送アクノリッジ（$\overline{\text{DTACK}}$）の同期がとられます．そして，ステート6（S 6）では，続くプロセッサの内部サイクルに対する要求がバス・インターフェース回路によって出力されます．ステート6（S 6）の終了時点〔ステート7（S 7）の開始時点〕で取り込まれたデータは，プロセッサの内部のデータ・バス・ホールド・レジスタにラッチされます．

ステート7（S7）において，上位データ・ストローブ（$\overline{\text{UDS}}$），下位データ・ストローブ（$\overline{\text{LDS}}$）は，ネゲートされています．アドレス・バス，アドレス・ストローブ（$\overline{\text{AS}}$），リード／ライト信号（R/$\overline{\text{W}}$），ファンクション・コード(FC 0～FC 2)は，サイクルのライト部分を実行するために状態を保持しています．スレーブ・デバイスは上位データ・ストローブ（$\overline{\text{UDS}}$）または下位データ・ストローブ（$\overline{\text{LDS}}$）のネゲーションを検出するまで，そのデータをアサート状態に保持します．スレーブ・デバイスは，データ・ストローブ（$\overline{\text{UDS}}$, $\overline{\text{LDS}}$）のネゲートの検出後1クロック以内にデータ転送アクノリッジ（$\overline{\text{DTACK}}$）とデータを取り除かなくてはなりません．ステート8（S 8）の間では何も新しい信号は出力されません．そして，この間内部では，データの修正が行われます．

図 4-9 リードーモディファイーライト・サイクルのフローチャート

BUS MASTER | **SLAVE**

Address Device (BUS MASTER)
1) Place Address on A1-A23
2) Set $R/\overline{W}$ to Read
3) Assert Address Strobe ($\overline{AS}$)
4) Assert Upper Data Strobe ($\overline{UDS}$) or Lower Data Strobe ($\overline{LDS}$)

↓

Input Data (SLAVE)
1) Decode Address
2) Place Data on D0-D7 or D8-D15
3) Assert Data Transfer Acknowledge ($\overline{DTACK}$)

↓

Acquire Data (BUS MASTER)
1) Latch Data
2) Negate $\overline{UDS}$ or $\overline{LDS}$
3) Start Data Modification

↓

Terminate Cycle (SLAVE)
1) Remove Data from D0-D7 or D8-D15
2) Negate DTACK

↓

Start Output Transfer (BUS MASTER)
1) Set $R/\overline{W}$ to Write
2) Place Data on D0-D7 or D8-D15
3) Assert Upper Data Strobe ($\overline{UDS}$) or Lower Data Strobe ($\overline{LDS}$)

↓

Input Data (SLAVE)
1) Store Data on D0-D7 or D8-D15
2) Assert Data Transfer Acknowledge ($\overline{DTACK}$)

↓

Terminate Output Transfer (BUS MASTER)
1) Negate $\overline{UDS}$ or $\overline{LDS}$
2) Negate $\overline{AS}$
3) Remove Data from D0-D7 or D8-D15
4) Set $R/\overline{W}$ to Read

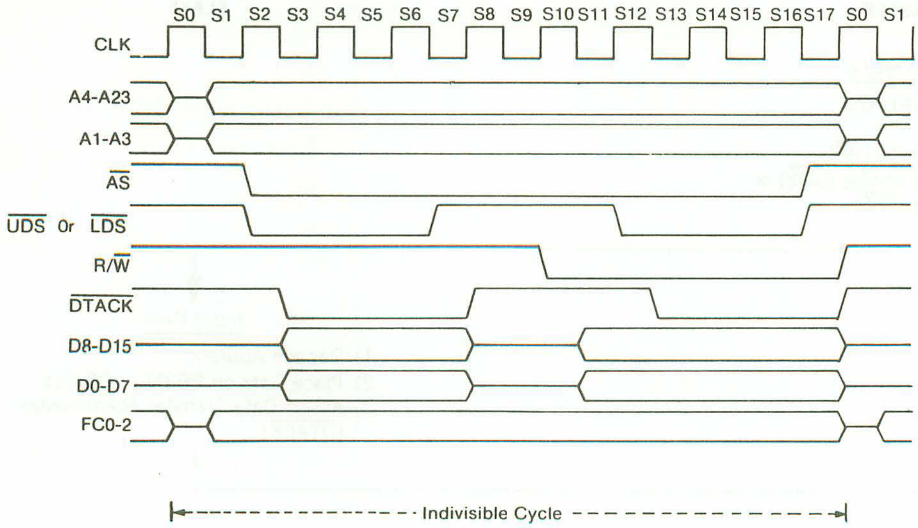
↓

Terminate Cycle (SLAVE)
1) Negate $\overline{DTACK}$

↓

Start Next Cycle (BUS MASTER)

図4-10 リード-モディファイ-ライト・サイクルのタイミング図



<注 意>

リード/ライト信号 (R/$\overline{\text{W}}$) は，バスと前のリード・サイクルの衝突を避けるためステート10 (S 10) まで"H"を保持します．そして，データ・バスは，ステート11 (S 11) になるまでプロセッサによりアサートされることはありません．

ステート9 (S 9) においては何も新しい信号は発生されません．

ステート10 (S 10) においてリード/ライト信号 (R/$\overline{\text{W}}$) は，ライト・サイクルを示すために"L"に変わります．外部データ・バス・バッファが必要な場合，リード/ライト信号 (R/$\overline{\text{W}}$) を使用して方向制御を行うことができます．この間のデータは，アサートとなっていないので（もし使用されている場合）外部のデータ・バッファに対して十分なターン・アラウンド・タイムを取ることが可能です．データはステート11 (S 11) でアサートされます．

ステート12 (S12) においてデータ・バスが定まったのを示すためにデータ・ストローブ ($\overline{\text{UDS}}$, $\overline{\text{LDS}}$) がアサートされます．スレーブ・デバイスはリード/ライト信号 (R/$\overline{\text{W}}$) とデータ・ストローブ ($\overline{\text{UDS}}$, $\overline{\text{LDS}}$) を使用し，データ・バスよりデータを取り込みます．スレーブ・デバイスは，そのデータをストアしたときデータ転送アクノリッジ ($\overline{\text{DTACK}}$) をアサートします．ステート13 (S 13) と14 (S 14) においては新しいコントロール信号は発生されません．

ステート15 (S 15) の開始時点までにデータ転送アクノリッジ ($\overline{\text{DTACK}}$) はアサート状態になっていなければなりません．そうでないとデータ転送アクノリッジ ($\overline{\text{DTACK}}$) がアサートされるまでプロセッサは待ち状態に入ってしまいます．ステート15 (S 15) で返答してきたデータ転送アクノリッジ ($\overline{\text{DTACK}}$) と同期が取られます．そして，ステート16 (S 16) でプロセッサ内部サイクルに対する要求がバス・インターフェース回路から出されます．

ステート17 (S 17) で，アドレス・ストローブ ($\overline{\text{AS}}$) とデータ・ストローブ ($\overline{\text{UDS}}$, $\overline{\text{LDS}}$) はネゲートされます．アドレス・バスとデータ・バスはステート17 (S 17) の間スタック・メモリ・オペレーションと信号のスキューが行えるように有効に保持されます．リード/ライト信号 (R/$\overline{\text{W}}$) とファンクション・コード (FC 0~FC 2) 出力は，転送オペレーションを確実に行うためステート17 (S 17) の間，有効に保持されます．スレーブ・デバイスは，アドレス・ストローブ($\overline{\text{AS}}$) またはデータ・ストローブ ($\overline{\text{UDS}}$, $\overline{\text{LDS}}$) のネゲーションを検出するまでデータ転送アクノリッ

ジ（$\overline{\text{DTACK}}$）をアサートしておきます．スレーブ・デバイスは，アドレス・ストローブ（$\overline{\text{AS}}$）またはデータ・ストローブ（$\overline{\text{UDS}}$, $\overline{\text{LDS}}$）のネゲーションを検出後，1クロック以内にデータ転送アクノリッジ（$\overline{\text{DTACK}}$）をネゲートしなくてはなりません．

注意；このためプロセッサは，ステート17（S17）の終了時点でデータ・バスを解放しますが，データ転送アクノリッジ（$\overline{\text{DTACK}}$）はステート0（S0）または，1（S1）まで解除されない可能性があります．その場合，アドレス・ストローブ（$\overline{\text{AS}}$）がネゲートされた時点でスレーブ・デバイスは解放されます．

(2) バス・アービトレーション

バス・アービトレーション（bus arbitration）は，マスタ・デバイスがバスのマスタ権（master ship）の要求（request），承認（granted），確認（acknowledge）を行う手法です．最も簡単な手続きとして次のようなものがあります．

(1) バス・マスタ権の要求をアサートする．

(2) 実行中のバス・サイクルが終了した時点で，バス使用の承認を受けとる．

(3) マスタ権の獲得を確認する．

図4-11に，1つのデバイスからの要求に対して行われるバス・アービトレーションのフローチャートを示します．図4-12には，その動作のタイミング図を示します．この手法によりデータ転送サイクル中に発生したバス・リクエスト（$\overline{\text{BR}}$）に対する処理が可能になります．

タイミング図に示されるように，バス・アクノリッジ（$\overline{\text{BGACK}}$）がアサートとなった時にバス・リクエスト（$\overline{\text{BR}}$）はネゲートされます．このタイプのオペレーションは，プロセッサとバス・マスタ権を獲得する能力のある1つのデバイスから構成されるシステムに対して適用できます．バス・マスタ権をもつことのできるデバイスを多く含むシステムの場合，各デバイスからのバス・リクエスト信号（$\overline{\text{BR}}$）はワイヤードORされてプロセッサに接続されます．このようなシステムでは，複数のバス・リクエストの発生がありえます．バス・アクノリッジ信号（$\overline{\text{BGACK}}$）がアサートされた後，数クロック後バス・グラント信号（$\overline{\text{BG}}$）がネゲートされるのが表に示されています．

しかしながら，まだ他のデバイスのバス・リクエスト（$\overline{\text{BR}}$）が残っていてバス・リクエスト（$\overline{\text{BR}}$）がアサートされたままでいる場合，プロセッサはバス・グラント（$\overline{\text{BG}}$）をネゲートし，その後，数サイクル以内に再びバス・グラント（$\overline{\text{BG}}$）をアサートします．これにより処理中のバス・マスタからの要求を完了する前に外部アービトレーション回路により次のバス・マスタを選択することが可能です．次にアービトレーション処理における各ステップについて説明します．

▩バスの要求法 バス・マスタになることのできる外部デバイスは，バス・リクエスト信号（$\overline{\text{BR}}$）をアサートすることによりバスを要求します．この信号は，ワイヤードORされた信号です（オープン・コレクタで構成する必要はありません）．そしてプロセッサに対し外部デバイスが外部バスのコントロール権を要求していることを示します．プロセッサは実際上外部デバイスよりも低いバス優先度にあり，プロセッサが実行中の最後のバス・サイクルが完了した後にバスを解放します．

バス・リクエスト信号（$\overline{\text{BR}}$）がアサートされていてバス・アクノリッジ（$\overline{\text{BGACK}}$）がアサートされていない時，プロセッサはバス・リクエスト（$\overline{\text{BR}}$）がネゲートとなると，自分の処理を続行します．このことにより，アービトレーション回路が誤って雑音に応答した場合でも通常の処理を続行できます．

図4-11　バス・アービトレーション・サイクルのフローチャート

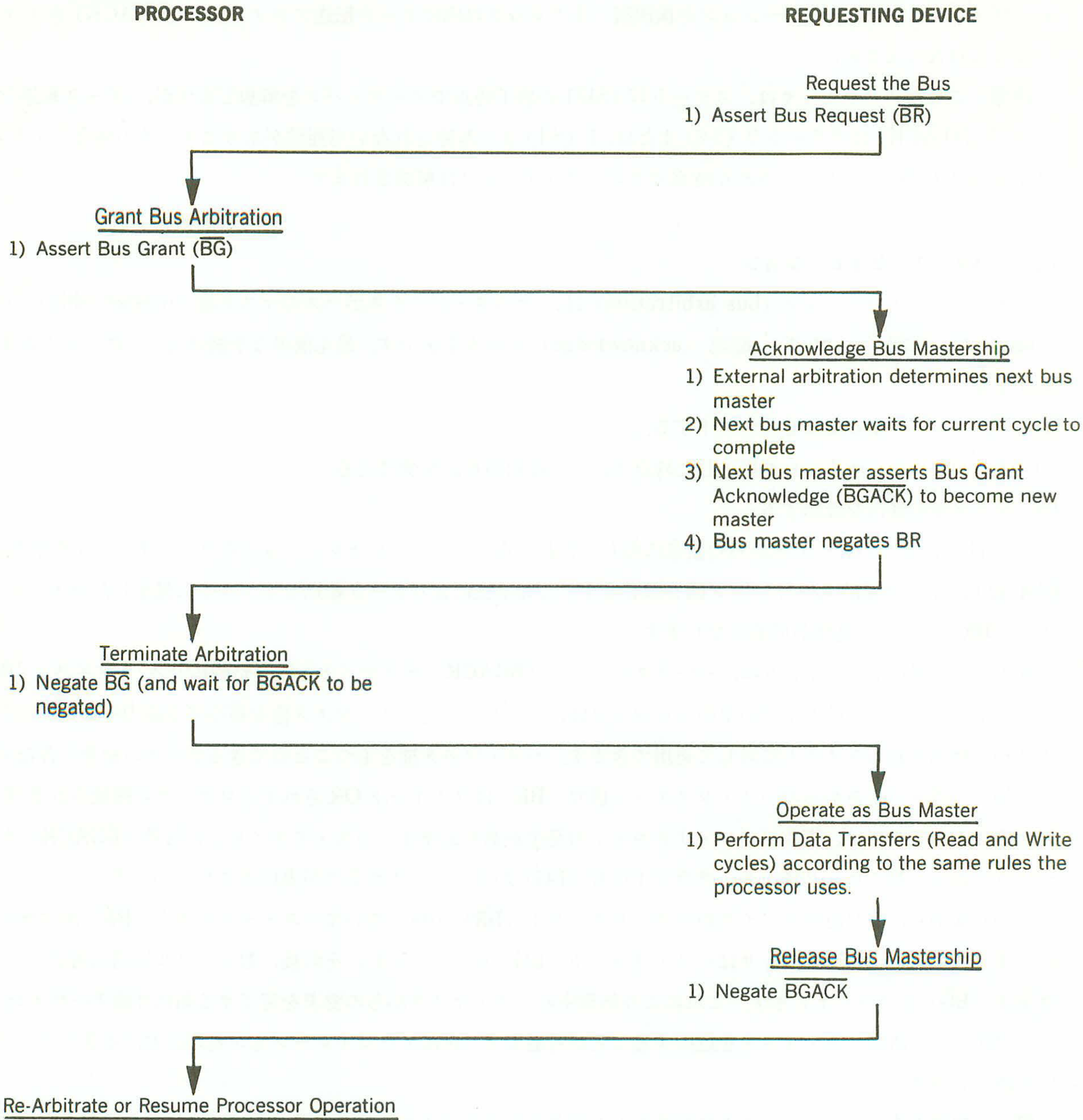
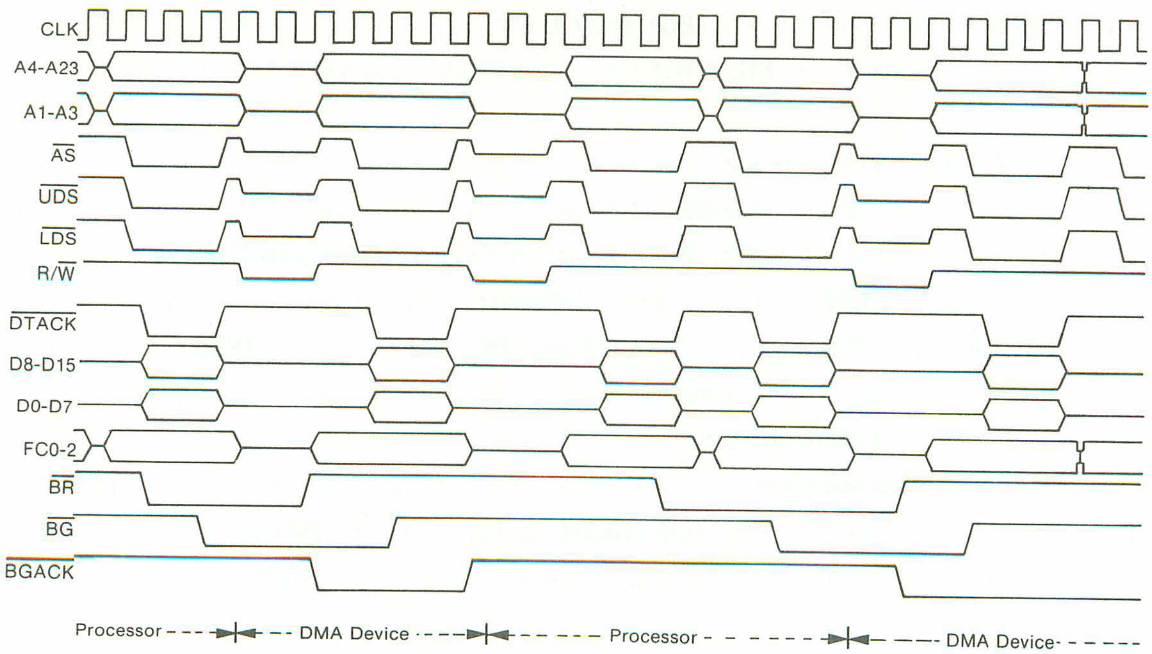


図4-12　バス・アービトレーション・サイクルのタイミング図



▩バス・グラントの受信　プロセッサはできるだけ早くバス・グラント（$\overline{BG}$）を出力します．通常この信号は内部同期の直後に発生します．これに対する唯一の例外として次の場合があります．すなわちプロセッサが，次のバス・サイクルを実行すると内部で決定したが，アドレス・ストローブ信号（$\overline{AS}$）をアサートにするのに充分なだけ，サイクルが進んでいない場合です．この場合バス・グラント（$\overline{BG}$）は，外部デバイスに対してバス・サイクルが実行中であることを示すために，アドレス・ストローブ（$\overline{AS}$）をアサートした後，1クロック後にアサートされます．

バス・グラント信号（$\overline{BG}$）は，daisy-chained network か priority-encoded network を通して伝えられます．プロセッサは，バス・アービトレーションのプロトコルが守られている限り外部アービトレーションの方法には影響されません．

▩マスタ権のアクノリッジ　バス・マスタ権を要求しているデバイスは，バス・グラント（$\overline{BG}$）を受信した後，現在バス・マスタとなっているデバイスのアドレス・ストローブ（$\overline{AS}$），データ転送アクノリッジ（$\overline{DTACK}$）とバス・グラント・アクノリッジ（$\overline{BGACK}$）がネゲートされるまで待ちます．その後，自分自身の $\overline{BGACK}$ を出力します．アドレス・ストローブ（$\overline{AS}$）のネゲートは前のバス・マスタがそのサイクルを完了したことを示します．バス・グラント・アクノリッジ（$\overline{BGACK}$）のネゲートは，前のマスタがバスを解放したことを示します（アドレス・ストローブ($\overline{AS}$)がアサートされている間，他のデバイスがサイクルに割り込むことはできません）．データ転送アクノリッジ($\overline{DTACK}$)のネゲートは前のスレーブ・デバイスと前のマスタとの間のデータ転送が完了したことを示します．

注意；データ転送アクノリッジ（$\overline{DTACK}$）がこのような機能をもたない応用システム（汎用デバイスがアドレス・ストローブのみに依存するように接続されているようなシステム）もあることがあります．バス・グラント・アクノリッジ（$\overline{BGACK}$）を発生してからネゲートするまでこのデバイスはバス・マスタとなります．バス・サイクルが完了してしまうまでバス・グラント・アクノリッジ（$\overline{BGACK}$）をネゲートしてはなりません．バス・マスタ権は，バス・グラント・アクノリッジ（$\overline{BGACK}$）をネゲートされると終了してしまうからです．

バス・マスタ権を認められたデバイスからのバス要求は，バス・グラント・アクノリッジ（$\overline{BGACK}$）をアサートした時，すぐにバス・リクエスト（$\overline{BR}$）をネゲートしなくてはなりません．もしバス・グラント・アクノリッジ

($\overline{BGACK}$) がネゲートされた後，バス・リクエスト ($\overline{BR}$) がアサートのままであったらプロセッサは別のアービトレーション処理を行い，別のバス・グラント ($\overline{BG}$) を出力します．

注意；プロセッサは，バス・グラント ($\overline{BG}$) を再びアサートしない限り，どの外部バス・サイクルも実行しません．

(3) バス・エラーとホルト・オペレーション

外部デバイスとハンドシェイクを行うことを前提とするバス・アーキテクチャにおいて，ハンドシェイクが行われなかったというエラーが生じることがあります．各システムはそれぞれ独自の最大応答時間に対処できるように，プロセッサはバス・エラー入力 ($\overline{BERR}$) をもっています．外部回路は，アドレス・ストローブ ($\overline{AS}$) とデータ転送アクノリッジ ($\overline{DTACK}$) 間の時間を測定し，エラーか正常かを判断してバス・エラー ($\overline{BERR}$) 信号を発生します．バス・エラー信号を受信すると，プロセッサはバス・エラー・エクセプション・シーケンスを開始するか，バス・サイクルをリラン（再実行）するかを選択します．

■例外シーケンス (exception sequence)　バス・エラー・エクセプション・シーケンス (bus error exception sequence) は，プロセッサがバス・エラー信号 ($\overline{BERR}$) を受信し，ホルト ($\overline{HALT}$) がネゲートである時に実行されます．図 4-13 にバス・エラー・エクセプション・シーケンスのタイミング図を示します．処理は次のような手順で行われます．

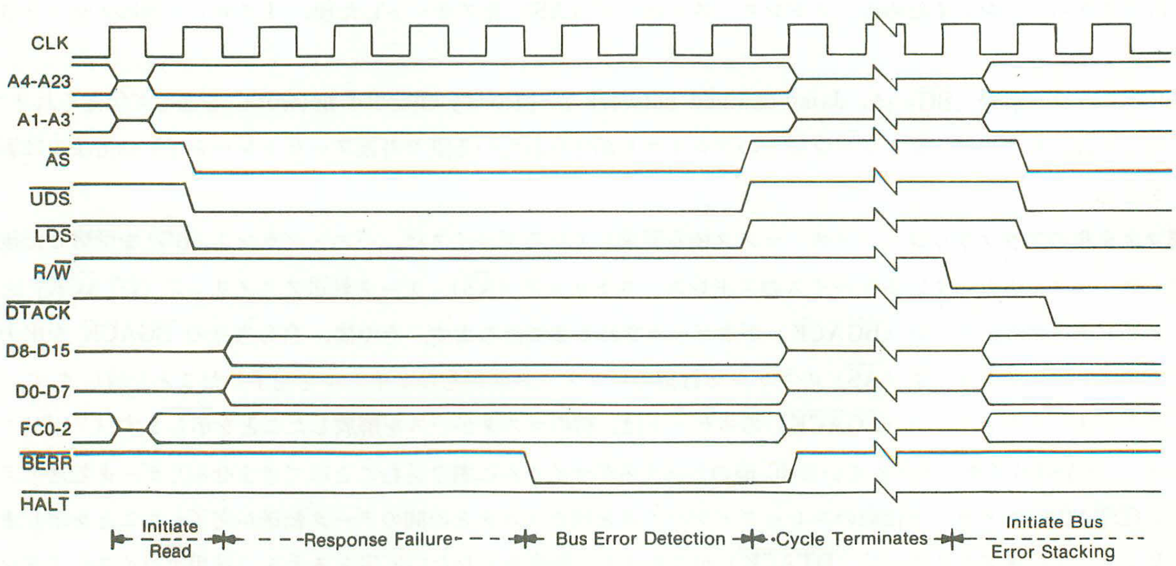
(1) プログラム・カウンタ (PC) とステータス・レジスタ (SR) をスタックする．

(2) エラー情報をスタックする．

(3) バス・エラー・ベクタ・テーブル・エントリを読む．

(4) バス・エラー・ハンドラ・ルーチンを実行する．

図 4-13 バス・エラーのタイミング図



プログラム・カウンタ (PC) とステータス・レジスタ (SR) のスタック動作は，割込みが生じた場合と同様に行われます．バス・エラー ($\overline{BERR}$) が生じた場合，その他のいくつかのデータもスタックされます．それらのデータは，エラーの性質を見極め，もし可能であれば修正して使用します．バス・エラー・ベクタはベクタ No. 2 でアドレス＄000008に格納されています．プロセッサは，このロケーションから新しいプログラム・カウンタ (PC) の値をロードします．バス・エラー・ハンドラ・ルーチン（ソフトウェア）は，この新しいプログラム・カウンタ (PC) の値から開始されます（第5章　エクセプション・プロセッシング　参照）．

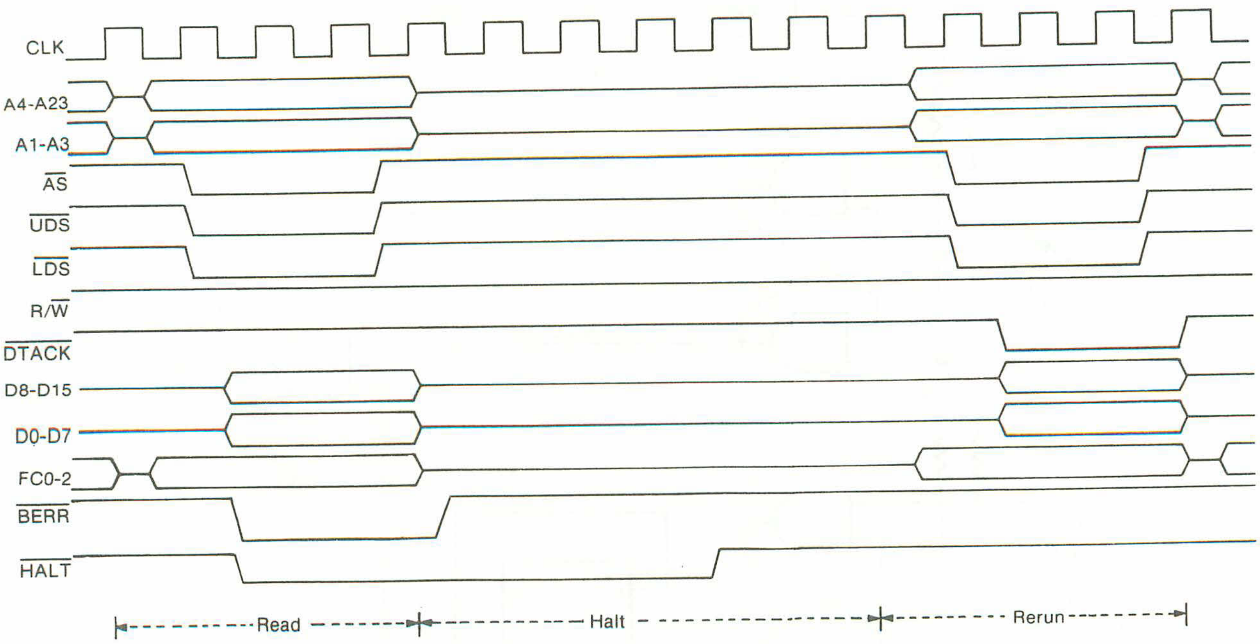
■バス・サイクルのリラン　プロセッサが，バス・エラー信号（$\overline{\text{BERR}}$）を受信し，ホルト端子が外部デバイスによってアサートされている場合，プロセッサはリラン・シーケンス（re-run sequence）を行います．図 4-14 にバス・サイクルのリラン・タイミング（re-run bus cycle timing）図を示します．

プロセッサは，バス・サイクルを完了した後，アドレス，データ，ファンクション・コード（FC 0～FC 2）そしてコントロール端子をハイ・インピーダンスにします．プロセッサは，ホルト状態になり，外部回路によってホルト信号（$\overline{\text{HALT}}$）がネゲートされるまで次のバス・サイクルは実行されません．ホルト信号（$\overline{\text{HALT}}$）がネゲートされるとプロセッサは，エラーを起こした時と同じアドレス，同じファンクション・コード，同じデータ（ライト・オペレーションの場合），同じコントロール信号を使用して，前と同じバス・サイクルをリランします．ホルト信号（$\overline{\text{HALT}}$）をネゲートする前にバス・エラー信号（$\overline{\text{BERR}}$）は，ネゲートされていなくてはなりません．

＜注　意＞

プロセッサは，リード-モディファイ-ライト・サイクルはリランしません．サイクル全体を正しく実行し TAS オペレーション中のライト・オペレーションが $\overline{\text{AS}}$ を解除することなく行われるように保証するためこの制限があります．

図 4-14　リランバス・サイクルのタイミング情報



■バス・エラーのない場合のホルト・オペレーション　MC 68000のホルト入力信号（$\overline{\text{HALT}}$）は，M 6800 のファミリにおけるホルト機能に類似した形でホルト（halt）／ラン（run）／シングル-ステップ（single-step）機能を行います．ホルト信号を適切な時間に変化させることにより，シングル-ステップ・モードを実行することができます．もしプロセッサに 1 ステップのバス・サイクルのみ実行させたい場合，プロセッサがバス・サイクルを実行し始めるまで“run”モードにしておき，その後“halt”モードにすることにより実現することができます．ユーザはシングル - ステップ・モードを使用し，プロセッサを 1 バス・サイクルずつ進めることができます（デバッグに使用できます）．

図 4-15 は，シングル - ステップ・オペレーションについてのタイミングを，図 4-16 にはシングル - ステップを行

図4-15　ホルト信号のタイミング特性

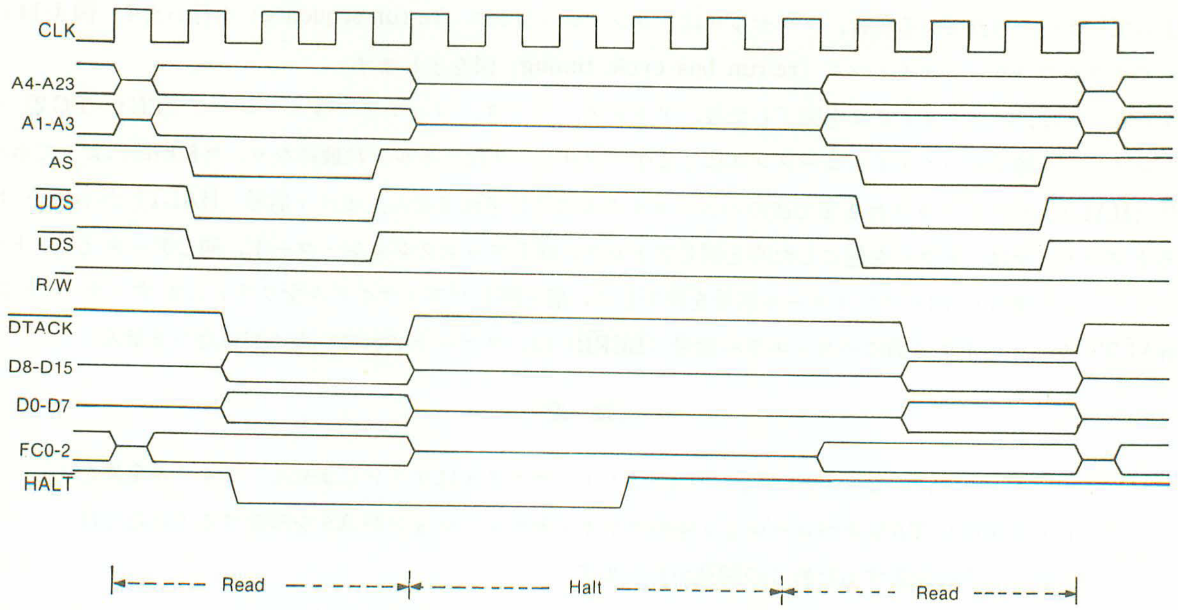
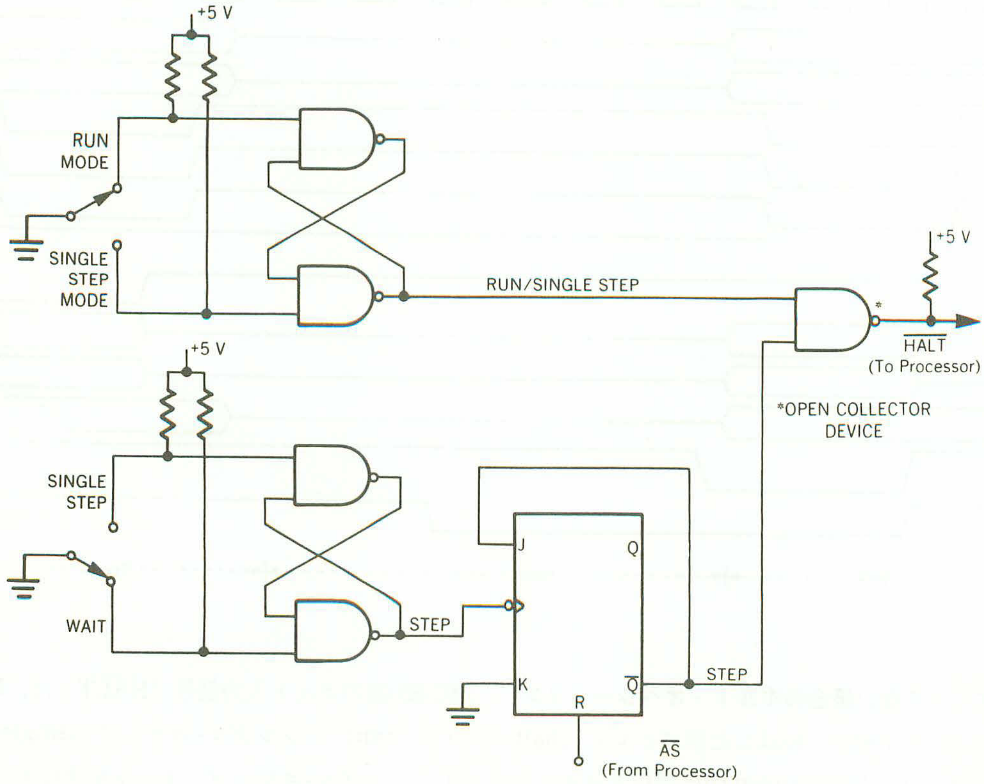


図4-16　単純化したシングル・ステップ回路



わせるための簡単な回路例を示します．シングル・ステップ・モードをデバッグ・ツールとして使用する場合，バス・エラー信号（$\overline{\text{BERR}}$）とホルト信号（$\overline{\text{HALT}}$）間における相互作用を避けるために多少の注意をしなければなりません．ホルト（$\overline{\text{HALT}}$）とリセット信号（$\overline{\text{RESET}}$）は，相互作用してシステムをリセットできるので，これらの信号間の相互作用についても同様の注意が必要です．

プロセッサは，ホルト信号（$\overline{\text{HALT}}$）がアサートされていることを検出した後，実行中のバス・サイクルを終了さ

せ，その後大部分の3ステート信号をハイ・インピーダンスにします．次のものがこれに含まれます．

(1) アドレス・ライン

(2) データ・ライン

(3) ファンクション・コード・ライン

このことは前項に書かれているバス・リラン・サイクル・オペレーション（re-run bus cycle）を正しく行うのに必要となります．

注意；プロセッサがホルト要求（$\overline{\text{HALT}}$）を受信した場合，ファンクション・コード（FC 0～FC 2）は，ハイ・インピーダンス状態となります（このバッファ特性はアドレス・バッファと同じです）．プロセッサがホルト要求を受け付けている間，バス・アービトレーションは通常と同じ動作をします．つまりホルトはバス・アービトレーションに何ら影響を与えません．

ホルト機能とハードウェアによるトレース機能を使用することにより，ハードウェア・デバッガで1バス・ステップあるいは1命令ずつトレースすることができます．ソフトウェア，デバッグ機能とこれらの機能によって全体的にフレキシビリティのある高いデバッグ能力が与えられています．

■ダブル・バス・フォルト　バス・エラー・エクセプション（bus error exception）が発生するとプロセッサはマシン状態についての情報をもっている数ワードをスタックしようとします．もしバス・エラーがこのスタック・オペレーション中に生じた場合，これは連続して2個のバス・エラーが生じたことになります．これはダブル・バス・フォルトと呼ばれ，ダブル・バス・フォルトが発生するとプロセッサはホルトします．ひとたびバス・エラー・エクセプションが発生すると，次の命令を実行する以前に起きるバス・エラーは，すべてダブル・バス・フォルトとなります．

注意；リランされたバス・サイクルは，バス・エラー・エクセプションとはならず，またダブル・バス・フォルトにもなりません．つまり外部ハードウェアがリランを要求している間は，プロセッサは継続して同じバス・サイクルをリランし続けます．

バス・エラー端子は，プロセッサが外部からリセット入力を受信した後もプロセッサのオペレーションに影響を与えます．後述のリセット・オペレーションの項で示されるように，プロセッサはリセットの後ベクタ・テーブルを読みプログラム実行開始アドレスを決定します．このベクタ・テーブルを読んでいる間に（または最初の命令実行前のどの時点においても）バス・エラーが発生した場合，プロセッサはダブル・バス・フォルトが発生した時と同様に応答しホルトします．ホルトしているプロセッサは外部からのリセット（$\overline{\text{RESET}}$）によってのみ新たに実行を開始することができます．

（4） リセット・オペレーション

リセット信号（$\overline{\text{RESET}}$）は，双方向性信号でプロセッサまたは外部からの信号によりシステムをリセットすることができます．図4-17にリセット・オペレーションのタイミング（reset operation timing）図を示します．プロセッサを確実にリセットするためには，ホルトとリセット・ラインを同時に“L”にしなければなりません．

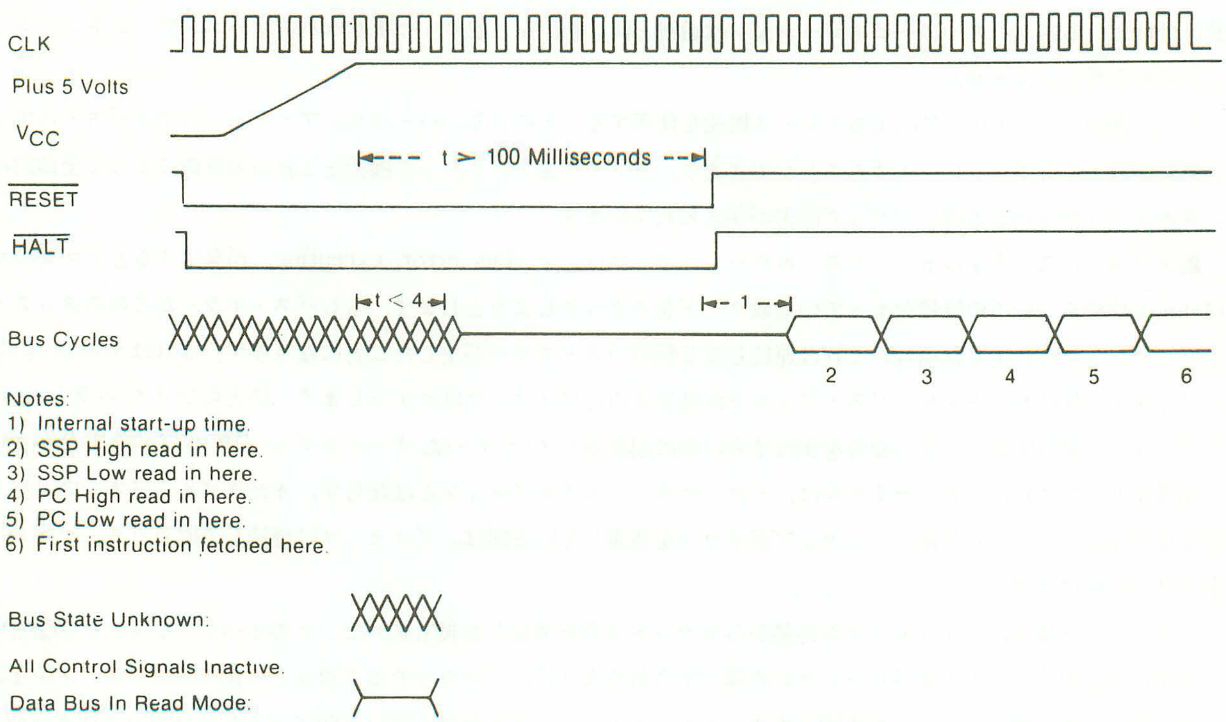
外部のデバイスによってリセットとホルト・ラインが“L”に駆動された場合，プロセッサを含めたシステム全体のリセットとみなされます．プロセッサはこれに対する応答としてリセット解除後，リセット・ベクタ・エントリ・テーブル（ベクタ番号，アドレスは＄000000）を読み，それをスーパバイザ・スタック・ポインタ（SSP）にロードします．次にベクタ・テーブル・エントリ番号1でアドレス＄000004のベクタを読み，プログラム・カウンタ（PC）にロードします．プロセッサはステータス・レジスタの割込みマスクを割込みレベル7に初期設定（initializes）します．リセット・シーケンスによって他のレジスタは一切の影響を受けません．

リセット命令が実行されると，リセット端子は出力となりプロセッサは124クロックの間，リセット端子を“L”に

します．この場合プロセッサは，自分自身以外のシステムの他の部分をリセットしようとしていることになります．このため，プロセッサの内部状態は影響を受けません．プロセッサの内部レジスタやステータス・レジスタは，リセット命令の実行によって影響されません．リセットラインに接続されている外部のデバイスはリセット命令により，リセットされていなくてはなりません．

初めて電源（V_{cc}）が加えられる場合，外部からのリセット信号は，V_{cc} が確定してから，少なくとも 100 ms の間“L”でなくてはなりません．

図 4-17　リセット・オペレーションのタイミング図



第5章　エクセプション・プロセッシング (例外処理)

本章では，MC 68000 の命令実行に関するノーマル・プロセッシング以外の処理について説明をします．また，ステータス・レジスタのスーパバイザ部の各ビット，すなわち，スーパバイザ／ユーザ・ビット，トレース・イネーブル・ビット，プロセッサ・プライオリティ・マスク・ビットの機能について説明します．さらに最後に，エクセプション・プロセッシングの時の，メモリ・レファレンスおよび，プロセッサの動作について詳述します．

5.1　処理状態 (Processing States)

MC 68000 は，常に，ノーマル (normal)，エクセプション (exception)，およびホルト(halt)の3種の処理状態のうち，いずれかの状態にあります．ノーマル・プロセッシングは，命令実行に関するもので，メモリ・レファレンスによって，命令および，オペランドをフェッチし，その結果をストアする処理です．ノーマル状態のうち，特殊な場合として，STOP 命令実行によって起きるストップ状態があります．STOP 命令が実行されると，プロセッサは，ストップ状態 (Stop state) になります．この状態においては，それ以後のメモリ・レファレンスは行われません．

エクセプション・プロセッシングは，割込み (interrupt)，トラップ (trap) 命令，および，トレースやその他の例外的な処理に関するものであり，命令や，命令実行中に生じる異常条件により，プロセッサ内部で発生するものや，また，外部から割込み，バス・エラー，およびリセットなどにより引き起こされるものがあります．エクセプション・プロセッシングは，プロセッサが異常条件に対して効率よく処理の流れを切り換えられるように設計されています．

ホルト状態 (Halt state) は，致命的なハードウェア上の異常により発生します．たとえば，バス･エラーのエクセプション・プロセッシング中に，さらに別のバス・エラーが発生すると，プロセッサはシステムを使用不能とみなし，ホルト状態となります．この状態にあるプロセッサをリスタートできるのは外部からのリセット ($\overline{\text{RESET}}$) のみです．注意しなければならないのは，ストップとホルトは，まったく違うものだということです．

5.2　特権状態 (Privilege States)

プロセッサは，“ユーザ”状態，もしくは“スーパバイザ”状態の2種の特権状態のうちのいずれかで動作します．特権状態は，どのオペレーションで処理を行うかを決定し，外部メモリ管理ユニット (MMU) によってアクセスをコントロールしたり，アドレス変換するために使用されます．また命令レファレンスにおいて，スーパバイザ・スタック・ポインタ (SSP) とユーザ・スタック・ポインタ (USP) のどちらを使用するかの選択にも用いられます．

特権状態は，コンピュータ・システムに安全性をもたせるためのものです．プログラムは，そのプログラム自身のコードとデータ領域のみをアクセスし，アクセスする必要のない領域，もしくは，変更してはならない領域までをもアクセスすることのないようにしなければなりません．

この特権機構は，大部分のプログラムをユーザ状態で実行することにより，安全性を保つことができます．ユーザ状態では，アクセスはコントロールされますので，システムの他の領域へのアクセスは許されません．オペレーティング・システム (OS) はスーパバイザ状態で動作し，すべてのリソースをアクセスでき，ユーザ状態のプログラムを管理するタスクを実行します．

(1) スーパバイザ状態 (Supervisor State)

スーパバイザ状態は特権状態の中でも上位に位置するものです．スーパバイザ状態は，ステータス・レジスタのSビットで判定され，Sビットが，アサート（論理真：1）の時，プロセッサはスーパバイザ状態で命令を実行します．スーパバイザ状態において，命令を実行した時に発生するバス・サイクルは，スーパバイザ・レファレンスとして分類されます．プロセッサが，スーパバイザ状態にある時には，システム・スタック・ポインタ，もしくは，アドレス・レジスタ7（A7）を使う命令は スーパバイザ・スタック・ポインタ（SSP）をアクセスすることになります．

すべてのエクセプション・プロセッシングは，Sビットの状態にかかわらず，スーパバイザ状態で実行されます．また，エクセプション・プロセッシング中におけるバス・サイクルは，スーパバイザ・レファレンスとして分類され，すべてのスタック操作には，スーパバイザ・スタック・ポインタ（SSP）が用いられます．

(2) ユーザ状態 (User State)

ユーザ状態は，特権状態の中でも下位に位置します．ユーザ状態は，ステータス・レジスタのSビットで判定され，Sビットが，ネゲート（論理負：0）の時，プロセッサはユーザ状態で命令を実行します．

ほとんどの命令は，ユーザ状態でも，スーパバイザ状態でも同様に実行できます．しかし，システムに対し重要な影響を与えるいくつかの命令は，スーパバイザ状態でしか実行できません．例えば，STOP命令やRESET命令は，ユーザ状態での実行を許されていません．また，ユーザ・プログラムからスーパバイザ状態には，ある制限された方式でしか入れないように，ステータス・レジスタ全体を操作する命令もスーパバイザ状態でしか実行できません．オペレーティング・システムとして使われるデバッギング・プログラムに有効な，ユーザ・スタック・ポインタへの移動命令（MOVE to USP）や，ユーザ・スタック・ポインタからの移動命令（MOVE from USP）は，ユーザ状態では実行できません．

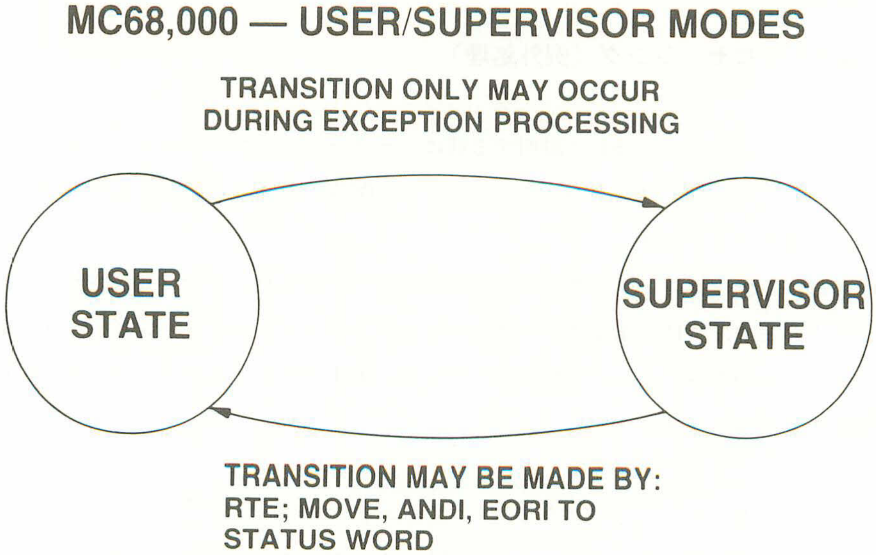
ユーザ状態において命令を実行した時に発生するバス・サイクルは，ユーザ・レファレンスとして分類されており，外部のメモリ管理デバイスがアドレス変換したり，アドレス空間のうちの保護されている部分のアクセスをコントロールしたりすることを可能にしています．プロセッサが，ユーザ状態にある間，システム・スタック・ポインタもしくはアドレス・レジスタ7（A7）を扱う命令は，ユーザ・スタック・ポインタ（USP）をアクセスします．

(3) 特権状態の変更 (Privilege State Changes)

プロセッサがユーザ状態で命令を実行し始めた後，特権状態を変更できるのはエクセプション・プロセッシングだけです．エクセプション・プロセッシングの間，ステータス・レジスタのSビットはセーブされ，その後にSビットがアサートされて，この処理はスーパバイザ状態となります．したがって，ユーザ状態の命令シーケンスを実行しているときエクセプション・プロセッシングが起き，命令実行を再開した場合，プロセッサはスーパバイザ状態となります．

スーパバイザ状態から，ユーザ状態への変更は，"return from exception"（RTE），"MOVE to status register"（MOVE word to SR），"AND immediate to status register"（ANDI word to SR），"exclusive OR immediate to status register"（EORI word to SR）の4命令のうち，いずれかによって行うことができます．RTE命令により，スーパバイザ・スタックから新しいステータス・レジスタとプログラム・カウンタをフェッチし，それらをそれぞれのレジスタにロードします．そして更新されたステータス・レジスタのSビットによって規定される特権状態の下で動作し，新しいプログラム・カウンタのアドレスから，命令のフェッチを開始します．MOVE/ANDI/EORI to SR 命令では，スーパバイザ状態で，それぞれのオペランドをフェッチし，その内容に従ってステータス

・レジスタは変更されます．そして更新されたステータス・レジスタのSビットによって規定される特権状態の下で動作し，次の継続するプログラム・カウンタのアドレスから，次の命令をフェッチします．



(4) レファレンス分類 (Reference Classification)

プロセッサの3本のファンクション・コード出力信号（FC 0，FC 1，FC 2）により，メモリ・レファレンスの種類を分類できます．つまり，アドレス，アクセス・コントロールおよび，割込みアクノリッジを外部で識別することができます．表5-1に，ファンクション・コード出力信号と，レファレンスの分類の関係を示します．

表5-1 レファレンス分類 (Reference Classification)

Function Code Output			Reference Class
FC2	FC1	FC0	
0	0	0	(Unassigned)
0	0	1	User Data
0	1	0	User Program
0	1	1	(Unassigned)
1	0	0	(Unassigned)
1	0	1	Supervisor Data
1	1	0	Supervisor Program
1	1	1	Interrupt Acknowledge

この表からも容易に理解できるように，ファンクション・コード出力信号を利用して，外部のメモリ空間をユーザ・プログラム，ユーザ・データ，スーパバイザ・プログラム，スーパバイザ・データの4つの空間に分類すれば，全体として最大64Mバイト（32Mワード）のメモリ空間をアクセスすることができます．

ファンクション・コードがプログラム・レファレンスを示すのは次の場合です．

(1) プログラム・カウンタがアドレス・ソースである場合．

(2) リセット・ベクタのフェッチ・サイクルである場合．

ファンクション・コードがデータ・レファレンスを示すのは次の場合です．

(1) アドレス・ソースがプログラム・カウンタ以外のオペランド・リード・サイクルである場合．

(2)　オペランド・ライト・サイクルである場合．

(3)　リセット以外のベクタ・フェッチ・サイクルである場合．

5.3　エクセプション・プロセッシング（例外処理）

割込み，トラップ，トレースについて詳しく説明する前に，エクセプション・プロセッシングの概略を説明します．エクセプション・プロセッシングは，そのエクセプション（例外）の起こる原因により（バリエーションはありますが）4つのステップをもっています．

まず第1ステップでは，ステータス・レジスタの一時的なコピーが作られ，そのコピーされたステータス・レジスタのSビットをアサートします．次に，第2ステップでエクセプション・プロセッシング・ベクタが決定され，第3ステップで現在のプロセッサの内容がセーブされます．そして，第4ステップではプロセッサの内容が一新され，プロセッサは命令処理に切り替わります．

(1)　エクセプション・ベクタ

エクセプション・ベクタとは，プロセッサが，エクセプション・プロセッシング・ルーチンのアドレスをフェッチするメモリ・ロケーションのことです．

エクセプション・ベクタは，4ワード長のリセット・ベクタを除き，すべて2ワード長です（図5-1）．リセット・ベクタは，スーパバイザ・プログラム・レファレンスで参照されますが，その他のエクセプション・ベクタは，すべてスーパバイザ・データ・レファレンスで参照されます．ベクタ番号は，8ビットで表されており，そのベクタ番号を4倍にすると，そのエクセプション・ベクタのアドレスが得られます．ベクタ番号は，エクセプションの起因によって，内部あるいは外部で作られます．割込みの場合，割込みアクノリッジ・バス・サイクル中（割込み処理中）に，周辺デバイスがデータ・ラインD0〜D7を使用して，8ビットのベクタ番号をプロセッサに送ります．図5-3に示すように，プロセッサはその8ビットのベクタ番号を，24ビットのアドレスに変換します．表5-2に，エクセプション・ベクタのメモリ・ロケーションを示します．

表5-2に示すように，メモリ・ロケーションは512ワード（1024バイト）長であり，アドレス0から1023が割り当

図5-1　エクセプション・ベクタのフォーマット

Word 0	New Program Counter (High)	A0 = 0, A1 = 0
Word 1	New Program Counter (Low)	A0 = 0, A1 = 1

図5-2　ベクタ番号のフォーマット

D15　　D8	D7							D0
Ignored	v7	v6	v5	v4	v3	v2	v1	v0

Where:
v7 is the MSB of the Vector Number
v0 is the LSB of the Vector Number

図5-3　8ビットのベクタ番号から交換されるアドレス

A23　　A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
All Zeroes	v7	v6	v5	v4	v3	v2	v1	v0	0	0

表5-2 エクセプション・ベクタの割り当て

Vector Number(s)	Address Dec	Address Hex	Address Space	Assignment
0	0	000	SP	Reset: Initial SSP
	4	004	SP	Reset: Initial PC
2	8	008	SD	Bus Error
3	12	00C	SD	Address Error
4	16	010	SD	Illegal Instruction
5	20	014	SD	Zero Divide
6	24	018	SD	CHK Instruction
7	28	01C	SD	TRAPV Instruction
8	32	020	SD	Privilege Violation
9	36	024	SD	Trace
10	40	028	SD	Line 1010 Emulator
11	44	02C	SD	Line 1111 Emulator
12*	48	030	SD	(Unassigned, reserved)
13*	52	034	SD	(Unassigned, reserved)
14*	56	038	SD	(Unassigned, reserved)
15*	60	03C	SD	(Uninitialized Interrupt Vector)
16-23*	64	040	SD	(Unassigned, reserved)
	95	05F		—
24	96	060	SD	Spurious Interrupt
25	100	064	SD	Level 1 Interrupt Autovector
26	104	068	SD	Level 2 Interrupt Autovector
27	108	06C	SD	Level 3 Interrupt Autovector
28	112	070	SD	Level 4 Interrupt Autovector
29	116	074	SD	Level 5 Interrupt Autovector
30	120	078	SD	Level 6 Interrupt Autovector
31	124	07C	SD	Level 7 Interrupt Autovector
32-47	128	080	SD	TRAP Instruction Vectors
	191	0BF		—
48-63*	192	0C0	SD	(Unassigned, reserved)
	255	0FF		—
64-255	256	100	SD	User Interrupt Vectors
	1023	3FF		—

＊ベクタ番号 12～23，48～63 は現在使用されていませんが，将来，使用される可能性があるため，ユーザの周辺デバイスにこれらの番号（アドレス）を割当てできません．

てられています．この中に，トラップ機能や，その他のシステム機能用に確保されているものを含めて，255 のユニークなベクタがあります．その 255 のうちベクタ番号 64 から 255 の 192 のベクタは，ユーザ割込みベクタ用に確保されています．ただし，最初の64のベクタ（0～63）は保護されていないため，ユーザ割込みベクタをシステム設計者の選択により重複させることができます．

（2） エクセプションの種類 (Kinds of Exception)

エクセプション（exception）は，内部あるいは外部の要因によって発生します．割込み，バス・エラー，リセットは外部に起因するエクセプションです．割込みは，周辺デバイスから出されるプロセッサへの動作要求であり，一方，バス・エラーとリセット入力は，アクセス・コントロールとプロセッサのリスタートのために使用されます．内部に起因するエクセプションとしては，命令によるもの，あるいはアドレス・エラーや，トレースによるものがあります．“trap”（TRAP），“trap on overflow”（TRAPV），“check register against bounds”（CHK），“divide”（DIV）命令はそれぞれの命令実行の一部としてエクセプションを発生し，さらに違法命令，および奇数アドレスからのワード・フェッチ（バウンダリ・エラー），特権違反もエクセプションを発生します．トレースは各命令の実行

後に，内部的に発生する優先度の高い割込みと同様の動作をします．

(3)　エクセプション・プロセッシング・シーケンス (Exception Processing Sequence)

エクセプション・プロセッシングには，4つのステップがあります．第1ステップでは，ステータス・レジスタが内部的にコピーされ，その後，Sビットがアサートされて，プロセッサは，スーパバイザ特権状態となります．また，Tビットはネゲートされるため，トレースによって妨害されることなく，エクセプション・プロセッシング・ルーチンが実行されます．リセットおよび割込みエクセプションでは，割込みプライオリティ・マスクも更新されます．

第2ステップではエクセプションのベクタ番号が決定され，割込みでは，ベクタ番号をプロセッサがフェッチします．これは，割込みアクノリッジ・レファレンスとして分類されています．その他のすべてのエクセプションでは，内部ロジックによりベクタ番号が決められており，このベクタ番号は・エクセプション・ベクタのアドレスを発生するのに使われます．

第3ステップでは，リセット・エクセプションの場合を除き，実行中のプロセッサの状態をセーブします．実行中のプログラム・カウンタの値とコピーされたステータス・レジスタの内容は，スーパバイザ・スタック・ポインタを用いてスタックされます．スタックされたプログラム・カウンタの値は通常，次のまだ実行されていない命令をさしています．しかし，バス・エラーとアドレス・エラーの場合には，スタックされたプログラム・カウンタの値はさだかでなく，多くの場合エラーの起こった命令のアドレスからインクリメントされています．またバス・エラー・エクセプションおよびアドレス・エラー・エクセプションの場合には，プロセッサがエラーを起こした時の実行中の内容を示す追加情報がスタックされます．

最後のステップは，すべてのエクセプションについて同一処理を行います．つまり，新しいプログラム・カウンタの値が，エクセプション・ベクタからフェッチされ，そのアドレスから実行すべき命令をフェッチしノーマルの命令・フェッチ，実行を再開します．

(4)　多重エクセプション (Multiple Exceptions)

この項では，同時に複数のエクセプションが重なって発生した場合の処理について説明します．エクセプションは，その発生要因，優先度 (priority) によって分類することができます（表5-3，グループ0～2参照)．

グループ0のエクセプションには，リセット，バス・エラー，アドレス・エラーがあり，これらのエクセプションが発生すると実行中の命令は排除され，プロセッサはただちにエクセプション・プロセッシングを開始します．グループ1のエクセプションには，トレース，割込み，特権違反，違法命令があります．これらのエクセプションが発生すると，現在実行中の命令が完結した後に，次の命令をプリフェッチ（先取り）し，エクセプション・プロセッシングを開始します（プログラム実行中，命令をプリフェッチした時に，特権違反や違法命令を検出します)．グループ2のエクセプションは，正規の命令処理の一部として発生します．TRAP，TRAPV，CHK，および“0”による除算のエクセプションは，このグループに属します．グループ2のエクセプションはノーマルの命令実行がエクセプション・プロセッシングにつながっています．

グループ0は，最も優先度の高いエクセプションであり，グループ2は，最も優先度の低いエクセプションです．また，グループ0の中でもリセットが最も高い優先度をもち，続いて，バス・エラー，アドレス・エラーの順となっています．同様に，グループ1の中では，トレースが割込みより高い優先度を持ち，割込みはまた，特権違反よりも優先度が高くなっています．グループ2では，一度に一つの命令しか実行できないため，優先順位はありません．

2つのエクセプションが同時に発生した場合，どちらを最初に実行するかは，優先順位により決定されます．たとえば，TRAP 命令の間に，バス・エラーが起こった場合は，バス・エラーガ優先されその後 TRAP 命令が実行され

MC68,000 — EXCEPTION PROCESSING SEQUENCE (NOT RESET)

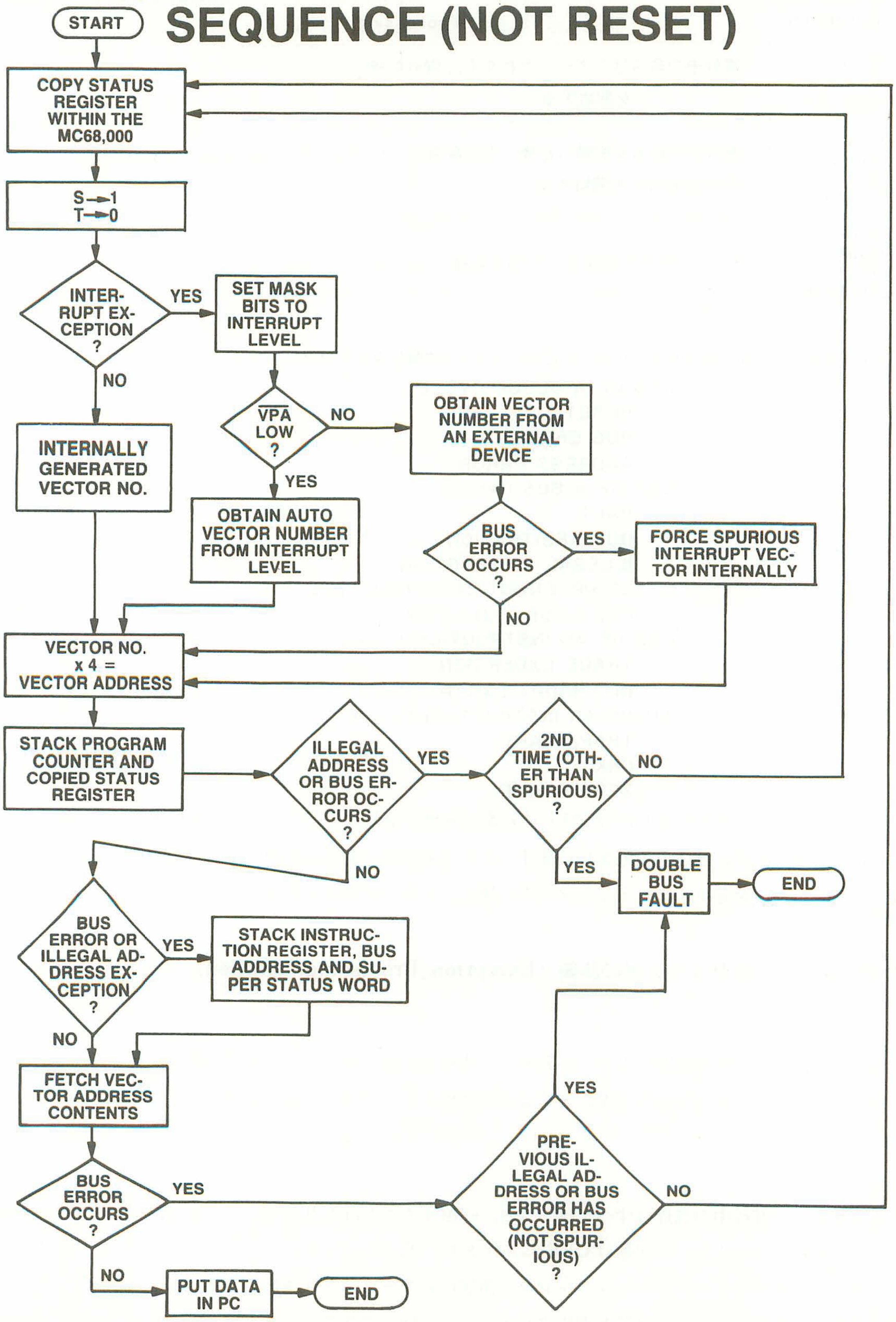


表5-3　エクセプション・プロセッシングの分類と優先度

Group	Exception	Processing
0	Reset Bus Error Address Error	実行中の命令はアボートされすぐに例外処理（Exception Processing）を開始する
1	Trace Interrupt Illegal Privilege	実行中の命令が完結した後，次の命令をプレフェッチし例外処理（Exception Processing）を開始する
2	TRAP, TRAPV, CHK, Zero Divide	ノーマルの命令処理として例外処理（Exception Processing）に入る

エクセプション・プロセッシング，ホールト，バス・アビトレーションの検出タイミング

- END OF A CLOCK CYCLE
  - RESET
  - BUS ERROR
  - ADDRESS ERROR
- END OF A BUS CYCLE
  - HALT
  - BUS ARBITRATION
  - ILLEGAL INSTRUCTION
  - UNIMPLEMENTED INSTRUCTION
  - PRIVILEGE VIOLATION
- END OF AN INSTRUCTION CYCLE
  - TRACE EXCEPTION
  - INTERRUPT EXCEPTIONS
- WITHIN AN INSTRUCTION CYCLE
  - TRAP, TRAPV
  - CHK
  - ZERO DIVIDE

ます．またTビットがアサートされており命令を実行中に割込み要求が起こるとトレースが優先され，そのままトレース処理を行い，完了後，次に，割込み処理を行います．そして最後に，割込み処理ルーチンの中で命令実行が再開されます．表5-3にエクセプション・プロセッシングの分類と，プライオリティを示します．

5.4　エクセプション・プロセッシングの詳細（Exception Processing Detailed）

エクセプションには，いくつかの原因があり，それぞれに特有の処理があります．以下の項で，エクセプションの原因，どのようにしてエクセプションとなるか，どのように処理するかについて述べます．

（1）　リセット

リセットが発生すると，進行中の処理はすべて排除され，回復することはありません．そして，プロセッサは強制的にスーパバイザ状態にされ，トレース状態はOFFにされます．割込みプライオリティ・マスクは，レベル7にセットされます．スーパバイザ・プログラム・レファレンス領域のロケーション0にあるリセット・エクセプション・ベクタを参照するために，内部でベクタ番号が発生されます．この時，プログラム・カウンタもステータス・レジスタもセーブされません．リセット・エクセプション・ベクタの最初の2ワードに格納されているベクタ・アドレスはスーパバイザ・スタック・ポインタの初期値としてフェッチされ，のこりの2ワードに格納されているベクタ・アドレ

スはプログラム・カウンタの初期値としてフェッチされます．その後，プログラム・カウンタで示されるアドレスから命令の実行を開始します．プログラム・カウンタの初期値は，パワー・アップ/リスタート・シーケンスの先頭アドレスを指定していなければなりません．

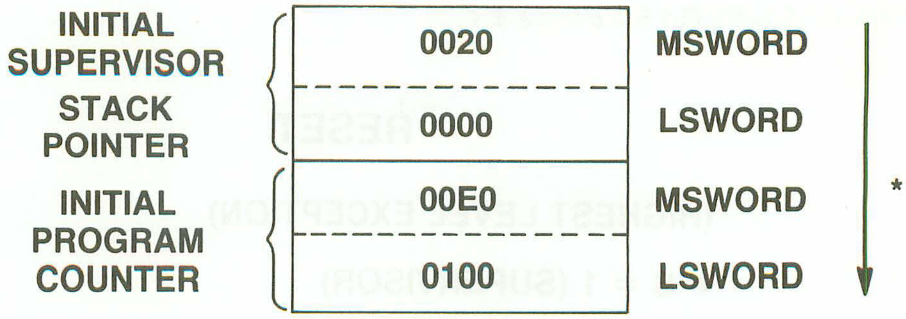
RESET 命令では，リセット・ベクタはロードされず，外部のデバイスをリセットするために，リセット・ラインがアサートされます．このようにソフトウェアによりシステムを所定の状態にリセットして，その後，RESET 命令の次の命令から処理を続けることができます．

RESET

(HIGHEST LEVEL EXCEPTION)

- **S = 1 (SUPERVISOR)**
- **T = 0 (TRACE OFF)**
- **INTERRUPT PRIORITY ⟶ LEVEL "7"**
- **RESET VECTOR CONTAINS SUPERVISOR STACK POINTER PLUS PROGRAM COUNTER**

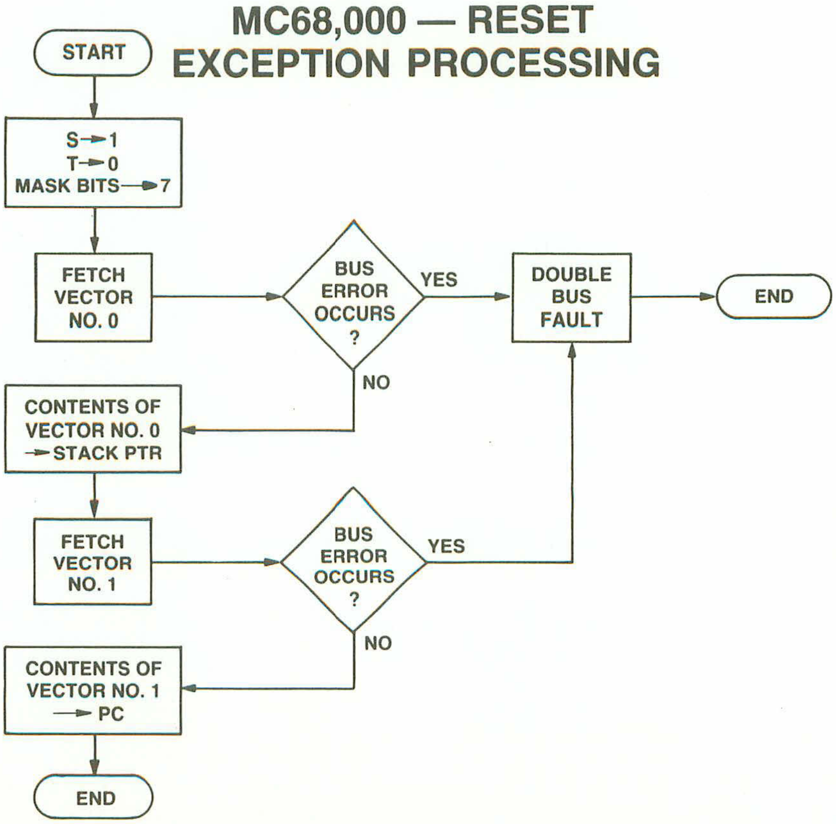
MC68,000 — RESET EXCEPTION VECTORS



INITIAL SSP = $00200000
INITIAL PC = $00E00100

*INCREASING ADDRESSES

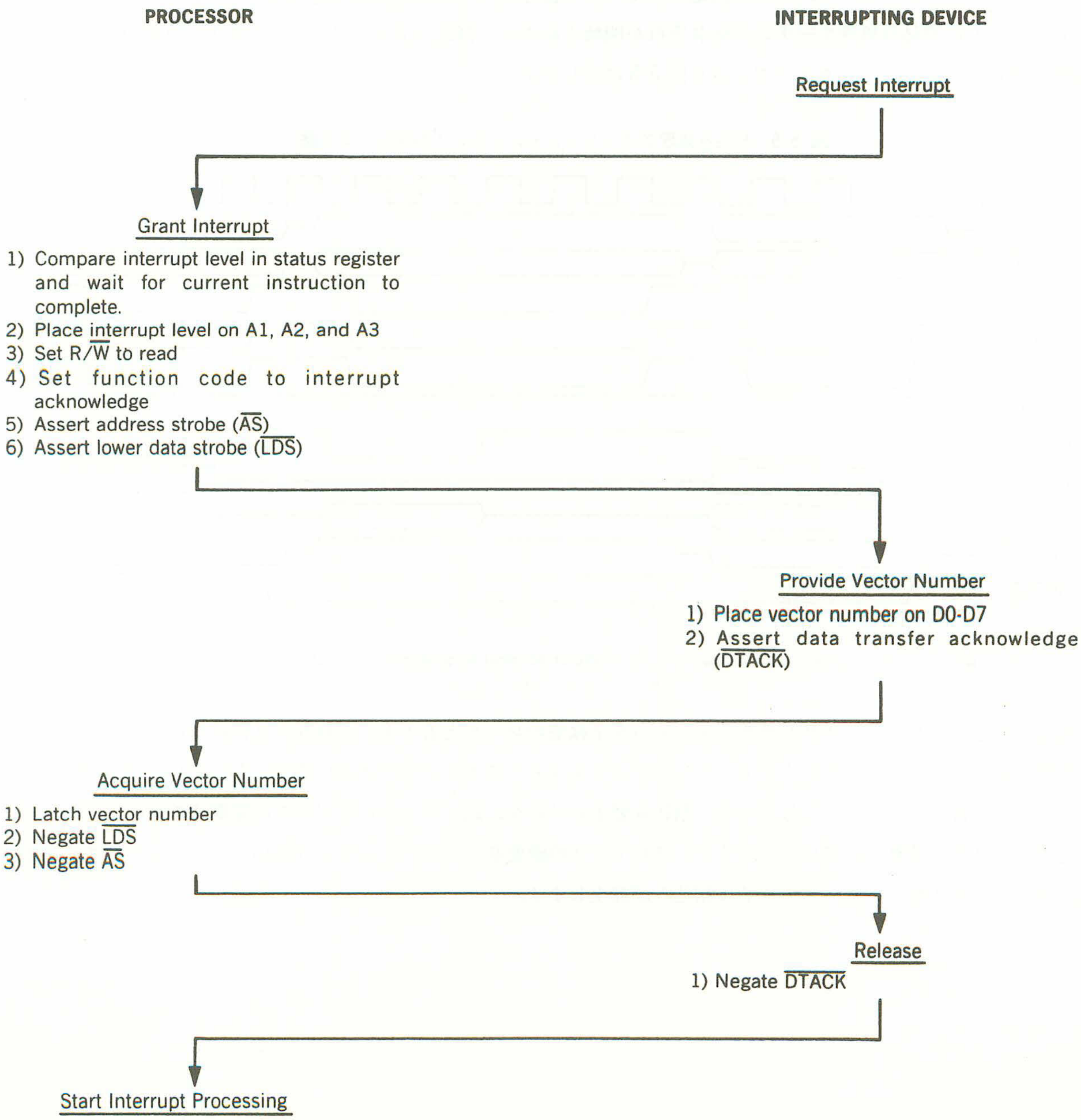
MC68,000 — RESET EXCEPTION PROCESSING



（2） 割込み（Interrupt）

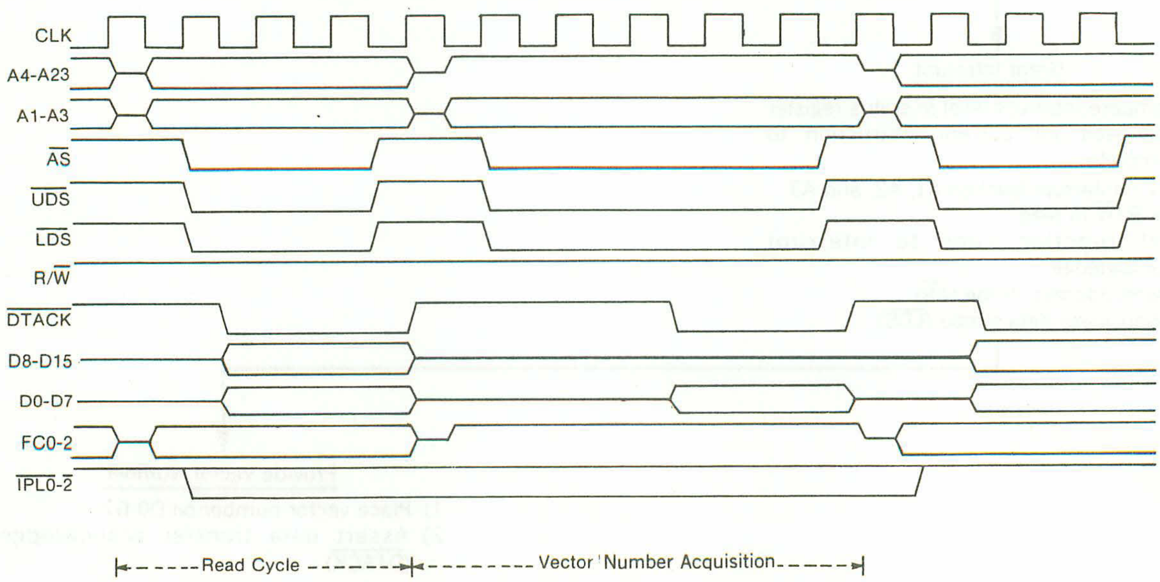
割込み優先度には7つのレベルがあり，周辺デバイスは，それぞれの割込み優先度レベルごとに接続されます．割込みを発生する周辺デバイスの数は制限されていません．割込み優先度レベルは1から7まで番号がつけられており，レベル7が最も優先度の高い割込みとなっています．ステータス・レジスタは，実行中の割込み優先度を示す3ビットの割込みマスクをもっており，実行中の割込み優先度以下のレベルの割込みは，すべて禁止されます．割込み・要求ライン（$\overline{\text{IPL}}$0，$\overline{\text{IPL}}$1，$\overline{\text{IPL}}$2）上の割込み要求レベルをエンコードすることにより，割込み要求がプロセッサに対して行われます（レベル0は，割込み要求がない，つまりノーマル状態を示します）．割込み要求がプロセッサに届いても即座にエクセプション・プロセッシングは行われず，実行中の命令が終了するまで待機させられます．その待機させられた割込みの優先度が実行中の割込み優先度以下の場合は，次の命令がそのまま実行されて割込みエクセプション・プロセッシングは延期されます（レベル7を検知した場合は，次項で説明するように，若干異なります）．

図 5-4 割込み処理アクノリッジ・シーケンスのフローチャート



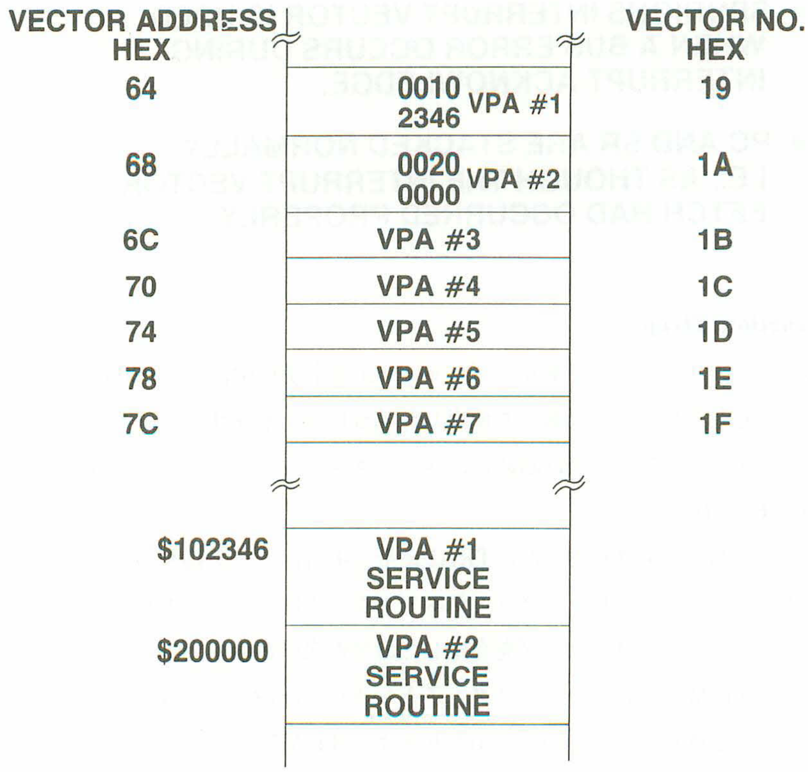
待機させられた割込みの優先度が実行中の割込み優先度よりも高い場合は，エクセプション・プロセッシング・シーケンスが開始されます．まず最初に，ステータス・レジスタのコピーがセーブされ，次にステータス・レジスタの特権状態は，スーパバイザにセットされて，トレースを禁止します．そして，さらに割込み優先度レベルは，現在受けつけられようとしている割込みのレベルにセットされます．プロセッサは，割込みデバイスからベクタ番号をフェッチします．これは，割込みアクノリッジ・レファレンスとして分類され，現在受けつけられようとしている割込みのレベルを，アドレス・バス(A 1，A 2，A 3)上に表示します．外部ロジックが，オート・ベクタを要求している場合には，割込みレベルによって決定されるベクタ番号がプロセッサ内部で発生されます．割込みアクノリッジの間，外部ロジックがバス・エラーを示している場合は，その割込みは，スプリアス（spurious）とみなされ，スプリアス割込みベクタを参照するためのベクタ番号が発生されます．その後，プロセッサはプログラム・カウンタとコピーされたステータス・レジスタをスーパバイザ・スタックにセーブして普通のエクセプション・プロセッシングを続けていきます．セーブされたプログラム・カウンタの値は割込みがなかった場合に実行されていたはずの命令のアドレスを示しています．それに続いて，すでに決定されたベクタ番号のベクタ・アドレスをフェッチし，プログラム・カウンタにロードし，割込み処理ルーチンの命令実行が開始されます．割込みアクノリッジ・シーケンスのフローチャートを図5-4に，タイミング・ダイアグラムを図5-5に示します．

図 5-5　割込み処理アクノリッジ・シーケンスのタイミング図

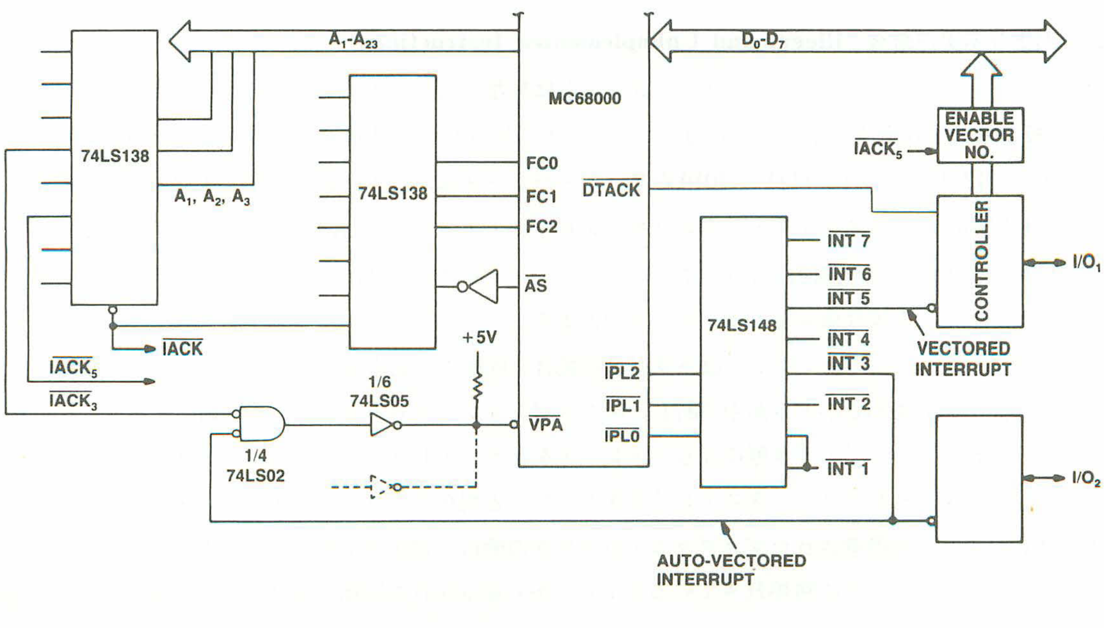


前にも少し述べましたが，プライオリティ・レベル7は他のレベルと若干違い，特殊な仕様になっています．レベル7の割込みは，割込みプライオリティ・マスクによってマスクすることはできないので，“ノンマスカブル割込み”としての能力を持っています．したがって，割込み要求レベルを，低レベルからレベル7に変更するたびに，割込み命令が発生します．**注意**；要求レベルが7で，プロセッサの優先度が命令によってより低いレベルにセットされた場合も，そのレベル比較により，レベル7の割込みが発生します．

MC68,000 — EXCEPTION VECTOR EXAMPLE



MC68,000 — VECTORED AND AUTO-VECTORED INTERRUPT LOGIC



MC68,000 — SPURIOUS INTERRUPT

- SPURIOUS INTERRUPT VECTOR IS USED WHEN A BUS ERROR OCCURS DURING INTERRUPT ACKNOWLEDGE.
- PC AND SR ARE STACKED NORMALLY, I.E., AS THOUGH THE INTERRUPT VECTOR FETCH HAD OCCURRED PROPERLY.

(3)　命令トラップ (Instruction Ttraps)

トラップは，命令に起因するエクセプションであり，プロセッサが命令実行中に異常な状態を認めるか，あるいは，通常動作としてトラップを発生させるような命令を使用した場合に発生します．

トラップのエクセプション・プロセッシングでは始めに，ステータス・レジスタがコピーされ，スーパバイザ状態になり，トレース状態は，OFF されます．

ベクタ番号は，プロセッサの内部で発生されますが，TRAP命令の場合，ベクタ番号の一部は，命令自身から与えられます．次に，ステータス・レジスタのコピーとプログラム・カウンタはスーパバイザ・スタックに退避されます．この退避されたプログラム・カウンタの値はトラップを発生した命令の次の命令のアドレスを示しています．そして最後に，エクセプション・ベクタに格納されているベクタ・アドレスから，命令の実行を開始します．

トラップ発生用として使用される特別な命令もいくつかあります．TRAP 命令は，常にエクセプションを発生させ，ユーザ・プログラムにとっては，システム・コールを実行するのに有用です．また，ユーザ・プログラムが算術演算オーバーフローやデータが指定されたビット長のサイズを超えるようなエラーを検出した場合，TRAPV 命令や CHK 命令はエクセプションを発生させます．

“Signed Divide” (DIVS) および “Unsigned Divide” (DIVU) 命令では，0の除数で除算すると例外を発生します．

(4)　違法命令および不実行命令 (Illegal and Unimplemented Instructions)

正規な命令のオペレーション・ワードのビット・パターンとは異なったビットパターンの命令を違法命令 (illegal instrnctions) と呼びます．命令実行中に，この命令がフェッチされると，違法命令エクセプションが発生します．

ビット15からビット12まで，つまり1111か1010のワード・パターンは，不実行命令として区別され，これらのパターンに，それぞれ個別にエクセプション・ベクタ（エミュレータ・トラップ）を割り当てることにより，効率よくエミュレーションができます．この機能により，オペレーティング・システムは，プログラム・エラーを検出したり，またはソフトウェアで，不実行命令をエミュレートできます．

違法命令のエクセプション・プロセッシングはトラップの場合と同様で，命令をフェッチしデコードした時に，プロセッサは，それが違法命令であるかどうかを判断し，エクセプション・プロセッシングを開始します．ステータス・レジスタがコピーされ，スーパバイザ状態になり，トレースをリセットします．その後，ベクタ番号を発生し，違法命令ベクタを参照します．そしてステータス・レジスタのコピーと現在のプログラム・カウンタがスーパバイザ・スタックに退避されます．この退避されたプログラム・カウンタの値は，違法命令のアドレスを示しています．そして最後に，エクセプション・ベクタに格納れさているアドレスから命令実行を開始します．不実行命令についても同様です．

(5) 特権違反 (Privilege Violations)

システムの安全性を確保するため，種々の特権命令があります．ユーザ状態で特権命令を実行しようとすると，エクセプションが発生します．特権命令には下記のものがあります．

STOP, RESET, RTE, MOVE to SR,
AND (word) Immediate to SR,
EOR (word) Immediate to SR,
OR (word) Immediate to SR,
MOVE USP

特権違反の場合のエクセプション・プロセッシングは，違法命令と類似しています．命令がフェッチしてデコードされた後に，プロセッサは，特権違反かどうかを判断し，エクセプション・プロセッシングを開始します．ステータス・レジスタはコピーされ，スーパバイザ状態に入り，トレースをリセットします．

次にベクタ番号を発生させ特権違反ベクタを参照し，ステータス・レジスタのコピーと，実行中のプログラム・カウンタを，スーパバイザ・スタックに退避させます．退避されたプログラム・カウンタの値は，特権違反を発生した命令のオペレーション・ワードのアドレスを示しています．そして最後に，特権違反のエクセプション・ベクタに格納されているアドレスから命令を実行します．

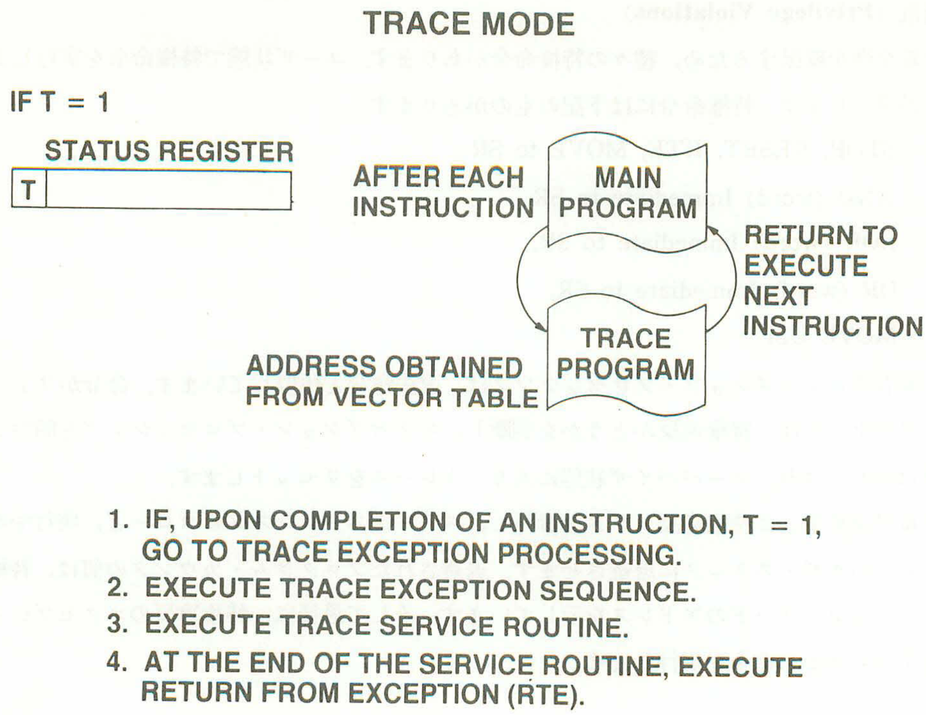
(6) トレース (Tracing)

プログラム開発上の助けとなるものとして，MC 68000 は命令ごとのトレースを可能とする機能を有しています．トレース状態では，各命令の実行後，エクセプションが発生します．これによりデバッグ・プログラムでテスト中のプログラムの実行をモニタすることが可能となります．

トレース状態では，ステータス・レジスタのスーパバイザ部のT (トレース) ビットを使用します．Tビットがネゲートの場合，トレースは行えずノーマルな命令を次々と実行します．T (トレース) ビットが命令実行を始めるときにアサートされている場合，命令実行終了後トレース・エクセプションを発生します．割込み受け付けのために命令が実行されなかった場合，または命令で違法命令や特権命令であった場合は，トレース・エクセプションは発生しません．リセット，バス・エラー，アドレス・エラー・エクセプションなどによって命令が中断される場合も，トレース・エクセプションは発生しません．命令が実際に実行され，割込みが命令終了まで待機させられている状態の場合には，割込みエクセプションより先にトレース・エクセプションが処理されます．

上記処理の代表的な例として，トレース状態で，TRAP 命令実行中に割込みが発生した場合を考えてみます．この場合は，最初にトラップ・エクセプションを処理し，次にトレース・エクセプションを，そして最後に割込みエクセプションを実行します．T (トレース) ビットがアサートで命令実行中に，その実行中の命令によりエクセプションが発生した場合，そのエクセプション・プロセッシングをトレース・エクセプションより先に行います．

トレース・エクセプション・プロセッシングの動作は，とても簡単になっています．命令実行終了後，次の命令を開始する前にエクセプション・プロセッシングが開始されます．ステータス・レジスタはコピーされ，スーパバイザ状態になります．そして，ステータス・レジスタのTビットが OFF となり，これ以上トレースはできなくなります．次にベクタ番号を作り，トレース・エクセプション・ベクタを参照し，ステータス・レジスタのコピーと現在のプログラム・カウンタをスーパバイザ・スタックにセーブさせます．そのセーブされたプログラム・カウンタの値は，次の命令のアドレスです．そして，トレース・エクセプション・ベクタに格納されているアドレスから命令実行を開始します．



(7) バス・エラー

外部ロジックからバス・エラーのエクセプション・プロセッシング要求がある場合，バス・エラー・エクセプションが発生し，プロセッサの現在のバス・サイクルは中断されます．プロセッサが命令を実行しているのかエクセプション・プロセッシングをしているかにかかわらず，その処理は終了させられてプロセッサは即時にエクセプション・プロセッシングを開始します．

バス・エラーのエクセプション・プロセッシングは，通常のエクセプション・プロセッシングの一連のステップに従って処理されます．ステータス・レジスタはコピーされ，スーパバイザ状態になり，トレース状態はOFFになります．次にベクタ番号を発生し，バス・エラー・ベクタを参照します．バス・エラー・エクセプション・プロセッシングが要求されたときには，プロセッサは所定の命令のところにはない（プロセッサの状態が，ノーマルの命令処理の状態にない）と考えられるので，プロセッサのもっと詳細な内容が必要となります．この多くの内容をセーブさせるため，追加情報をスーパバイザ・スタックにセーブします．もちろん，ステータス・レジスタのコピーとプログラム・カウンタもセーブされます．そのセーブされたプログラム・カウンタの値は，バス・エラーを発生した命令のオペレーション・ワードのアドレスより，数バイト先に進んでいるかもしれません．

また次の命令をフェッチしている間にバス・エラーが発生した場合は，たとえその命令が "branch"，"jump"，"return" 命令であったとしても，セーブされたプログラム・カウンタの値は現在の命令付近の値を示しているかもしれません．このため，通常の情報以外にプロセッサは処理中だった命令のオペレーション・ワードの内部コピー（図5-6の Instruction Register）と，中断されたバス・サイクルによってアクセスされつつあったアドレス（図5-6の Access Address）をセーブします．

さらにバス・エラー時のアクセスに関する次の特殊情報を退避します．

(1) リードとライト（R/W）のどちらかであるか．

(2) "命令実行中" かそうでない時（I/N）に起こったか．

(3) ファンクション・コード（Function Code）出力のメモリ・レファレンス分類は何であったか．

この "命令実行中" というのは，プロセッサが通常状態の命令を処理しているか，または表5-3のグループ2のエ

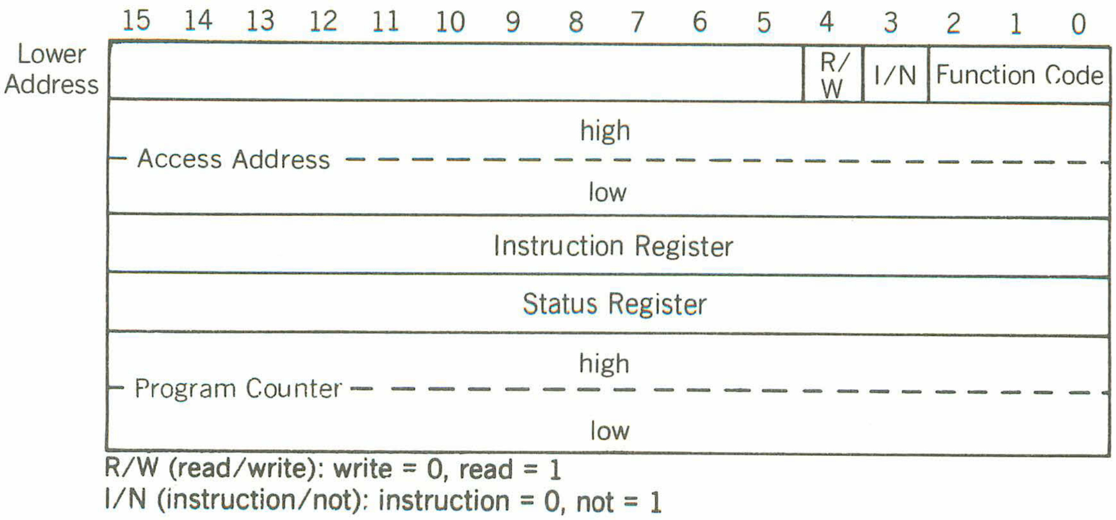
クセプション・プロセッシングをしていることを示します．表5-3のグループ0あるいは1のエクセプション・プロセッシングをしているとき，“命令処理実行中”には含めません．図5-6にスーパバイザ・スタックでこれらの情報がどのように構成されているか示します．

もしエクセプション・プロセッシングの最後の段階で（エクセプション・ベクタを読み込んでいる間，または命令をフェッチしている間に）バス・エラーが発生した場合，プログラム・カウンタの値はエクセプション・ベクタのアドレスになります．

一般的には，これらのデータだけでは十分にバス・エラーから回復することはできませんが，これによってソフトウェアの診断ができます．最後にプロセッサは，バス・エラー・エクセプション・ベクタに格納されているアドレスから命令処理を開始します．スタックを一掃して，どこまで実行を続けるのか判断するのは，エラー・ハンドラ・ルーチンの役割です．

バス・エラー，アドレス・エラー，またはリセットのエクセプション・プロセッシング中にバス・エラーが発生した場合，プロセッサはホルト（Halt）し，すべての処理も中止されます．プロセッサは全メモリ内容を破壊することなく，システムから切り離されるので，上記の処理によってこの重大なシステム上の障害を簡単に検出できます．外部からのリセット入力によってのみ，このホルトしたプロセッサを再動作させることができます．

図 5-6　バス・アドレス・エラー・エクセプションのためのスーパバイザ・スタック配列



（8） アドレス・エラー

プロセッサがワード・オペランド，ロング・ワード・オペランドあるいは，命令を奇数アドレスでアクセスしようとした場合，アドレス・エラー・エクセプションが発生します．これは内部起因のバス・エラーと同様の効果をもち，バス・サイクルを中断し，現在どの処理を実行しているかにかかわらずその処理を中止し，エクセプション・プロセッシングを開始します．エクセプション・プロセッシング開始後のシーケンスは，スタックされる情報も含め，バス・エラーのシーケンスと同様です．ただし，ベクタ番号はアドレス・エラーのベクタです．さらに，アドレス・エラーがバス・エラー，アドレス・エラー，リセットのエクセプション中に発生した場合は，プロセッサがホルトする点も同様です．

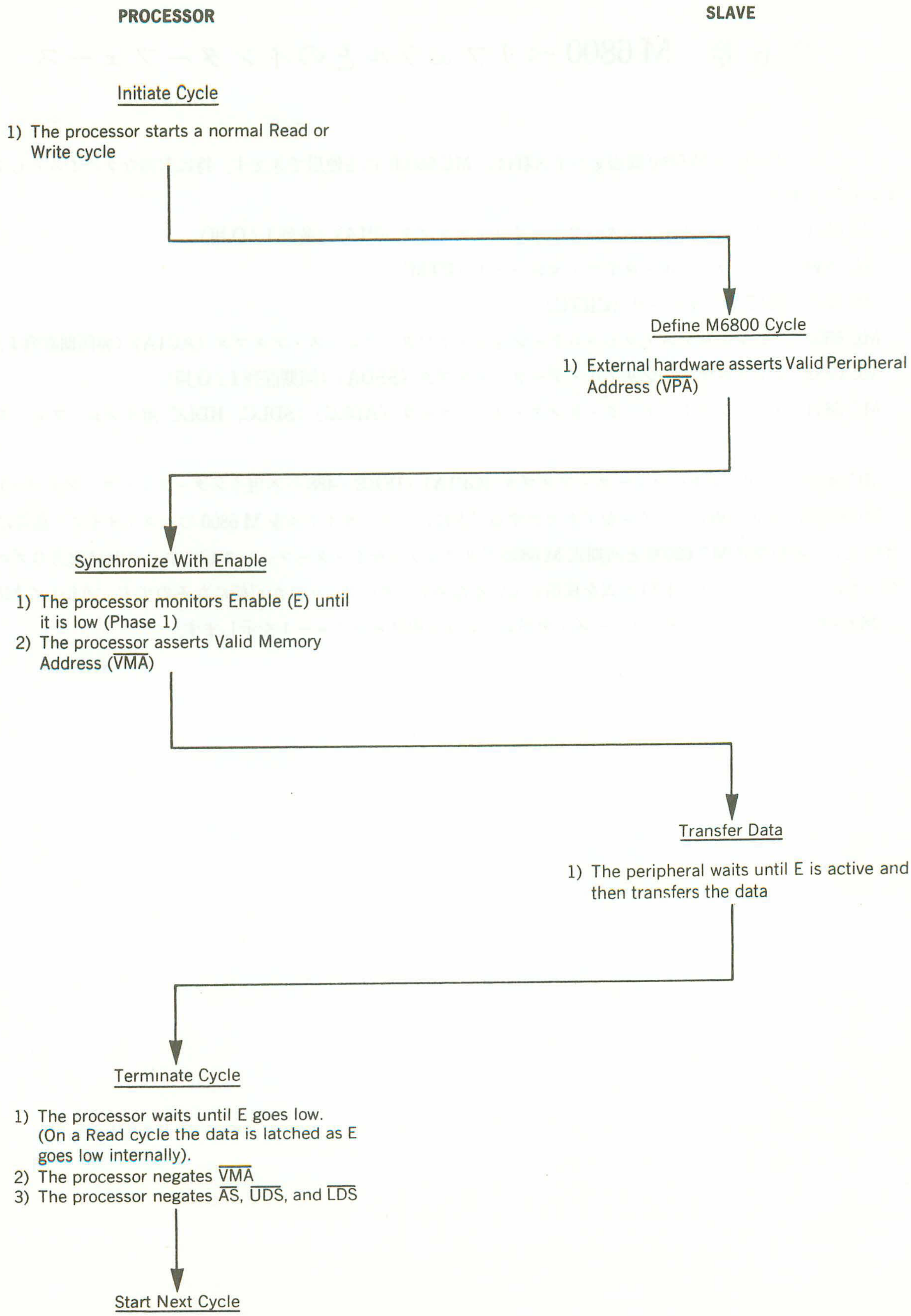
第 6 章　M 6800 ペリフェラルとのインターフェース

モトローラの幅広い M 6800 周辺デバイス群は，MC 68000 にも使用できます．特に有用なデバイスとして以下のものがあります．

MC 6821　ペリフェラル・インターフェーイス・アダプタ（PIA）（並列 I / O 用）
MC 6840　プログラマブル・タイマ・モジュール（PTM）
MC 6845　CRT コントローラ（CRTC）
MC 6850　アシンクロナス・コミュニケーション・インターフェース・アダプタ（ACIA）（非同期直列 I / O 用）
MC 6852　シンクロナス・シリアル・データ・アダプタ（SSDA）（同期直列 I / O 用）
MC 6854　アドバンスト・データ・リンク・コントローラ（ADLC）（SDLC，HDLC 用インターフェース・アダプタ）
MC 68488　汎用インターフェース・アダプタ（GPIA）（IEEE・488 バス用インターフェース・アダプタ）

MC 68000 は M 6800 デバイスをアクセスするごとに，バス・サイクルを M 6800 のバス・サイクル条件に合うよう修正し，非同期式 MC 68000 と同期式 M 6800 ペリフェラルをインターフェースします．これは両方のプロセッサがともにメモリ・マップド I/O 方式を採用しているためインターフェースが可能になるのです．図 6-1 に MC 68000 と M 6800 デバイスのインターフェース・オペレーションのフローチャートを示します．

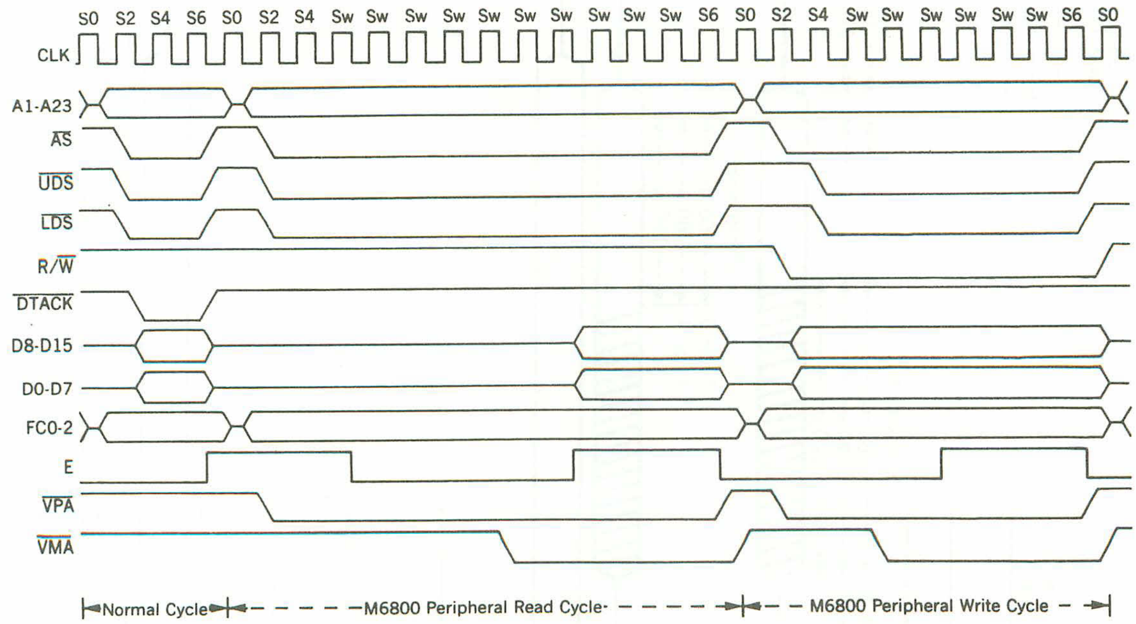
図 6-1 M 6800 インターフェースのフローチャート



6.1 データ転送オペレーション

M 6800 のサイクル・タイミングを図 6-2 で示します.

図 6-2 M 6800 のサイクル・オペレーション



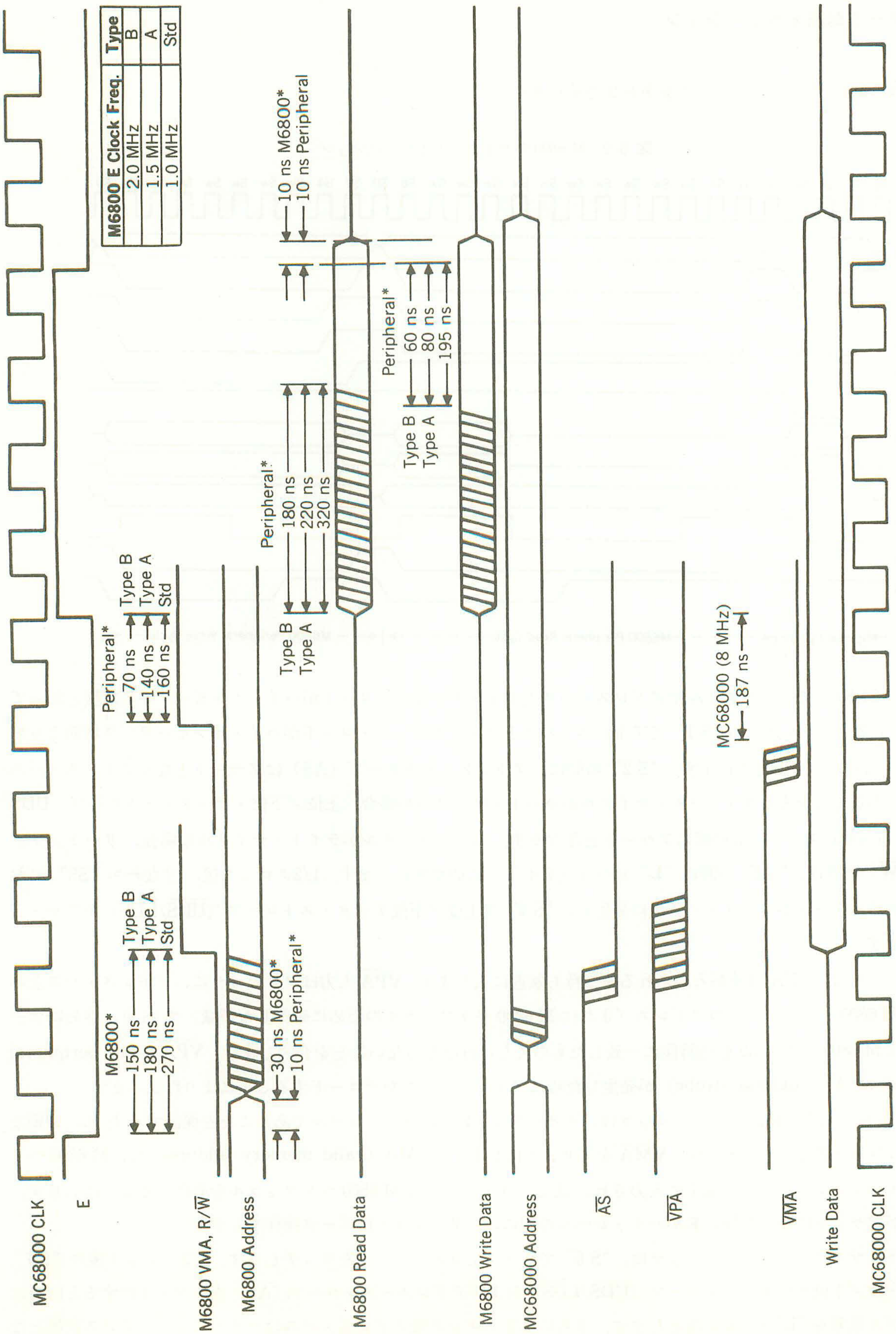
最初に CLK の“S 0”サイクルでアドレス・バスとファンクション・コードがハイ・インピーダンス状態となっています. 1/2 サイクル遅れの“S 1”でアドレス・バスとファンクション・コードがハイ・インピーダンス状態より解放され, 信号として出力されます. “S 2”の間に, アドレス・ストローブ (AS) はアサートとなりアドレス・バスが有効であることを表します. バス・サイクルがリード・サイクルの場合, 上位／下位・データ・ストローブ ($\overline{\mathrm{UDS}}$/$\overline{\mathrm{LDS}}$) も同様にステート 2 の間にアサートとなります. バス・サイクルがライト・サイクルの場合, リード／ライト (R/$\overline{\mathrm{W}}$) 信号は, “S 2”の間に“L”レベル (ライト) に切り替わります. 1/2クロック後, すなわち“S3”において書き込みデータがデータ・バス上に発生し, “S 4”で上位／下位データ・ストローブ ($\overline{\mathrm{UDS}}$/$\overline{\mathrm{LDS}}$) がアサートとなります.

プロセッサは, $\overline{\mathrm{VPA}}$ 信号が入力されるまで待ち状態に入ります. $\overline{\mathrm{VPA}}$ 入力はプロセッサに, アドレス・バス上の情報が M 6800 ペリフェラルのアドレス (または M 6800 ペリフェラルのために確保した領域) であり, さらにデータ操作は M 6800 バスの $\phi 2$ 転送特性に一致したものでなければならないことを合図します. $\overline{\mathrm{VPA}}$ (valid peripheral address) は $\overline{\mathrm{AS}}$ (address strobe) が発生した時にアドレス・バスをデコードすることにより作成します.

$\overline{\mathrm{VPA}}$ 信号の入力を確認したプロセッサは, イネーブル (E) がロー・レベルであることを保証するため, 必要な場合には待ち状態となり, その後, $\overline{\mathrm{VMA}}$ をアサートにします. VMA (valid memory address) は, M 6800 ペリフェラルのチップ・セレクト端子に入力され, 正しいタイミングで M 6800 ペリフェラルを動作するようにします. $\overline{\mathrm{VMA}}$ がアサートになった後, E がハイ・レベルの間に, ペリヘェラルはデータ操作をします.

リード・サイクルの時, プロセッサは, “S 6”でペリフェラル・データをラッチします. 1/2 クロック後の“S 7”の間に上位／下位データ・ストローブ ($\overline{\mathrm{UDS}}$/$\overline{\mathrm{LDS}}$) およびアドレス・ストローブ ($\overline{\mathrm{AS}}$) をネゲートにすると同時にイネーブル信号を“L”レベルに落とします. さらに 1/2 クロック後アドレス・バスはハイ・インピーダンス状態とな

図 6-3　M 6800周辺デバイスに対するMC 68000のタイミング



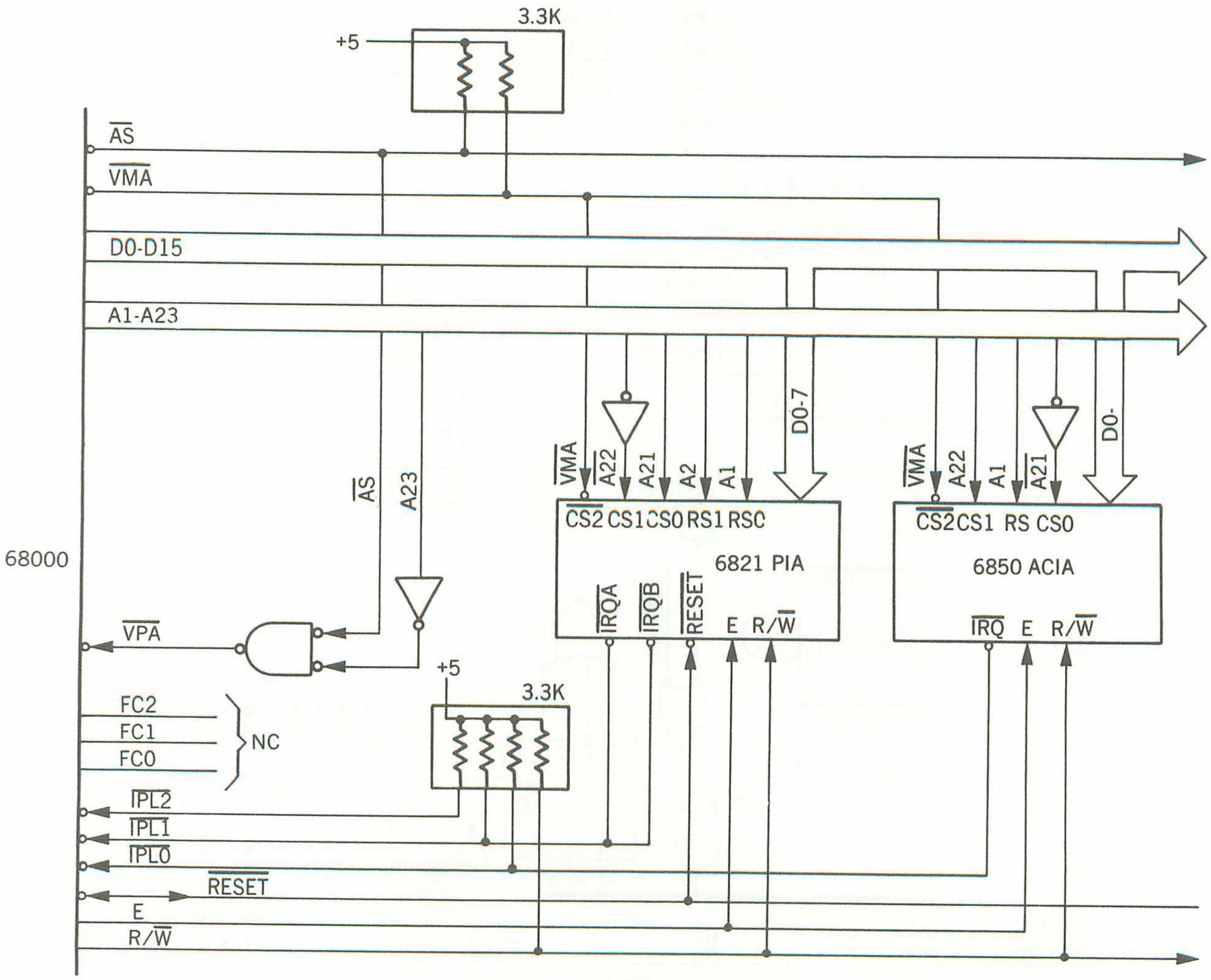
*Times are expressed for different device clock frequencies

ります．このサイクルの間データ・バスはハイ・インピーダンス状態となります．リード／ライト信号（R/$\overline{\mathrm{W}}$）はこの時“H”レベル（リード）に切り替わります．アドレス・ストローブがネゲートにされた後1クロック・サイクルの間にペリフェラル・ロジックは $\overline{\mathrm{VPA}}$ を取り去る必要があります．

図6-3にM6800用に規定され，かつM6800ペリフェラルが必要とするタイミング，それに対応するMC68000のタイミングを示します．ペリフェラルのタイミングに関する詳細は，モトローラの個々のペリフェラル・デバイス・データシートに記載されています．M6800のVMAがアクティブ・ハイであるのに対し，MC68000の $\overline{\mathrm{VMA}}$ は，アクティブ・ローであることに注意する必要があります．

図6-4，図6-5にM6800ペリフェラルとMC68000とのインターフェース例を示します．図6-4のシステムでは，上位8Mバイトのメモリを M6800ペリフェラル用にリザーブしてあります．図6-5のシステムは，メモリ・マッピングの効率がより優れ容易に拡張できますがやや複雑です．

図 6-4　M6800のインターフェース（例1）



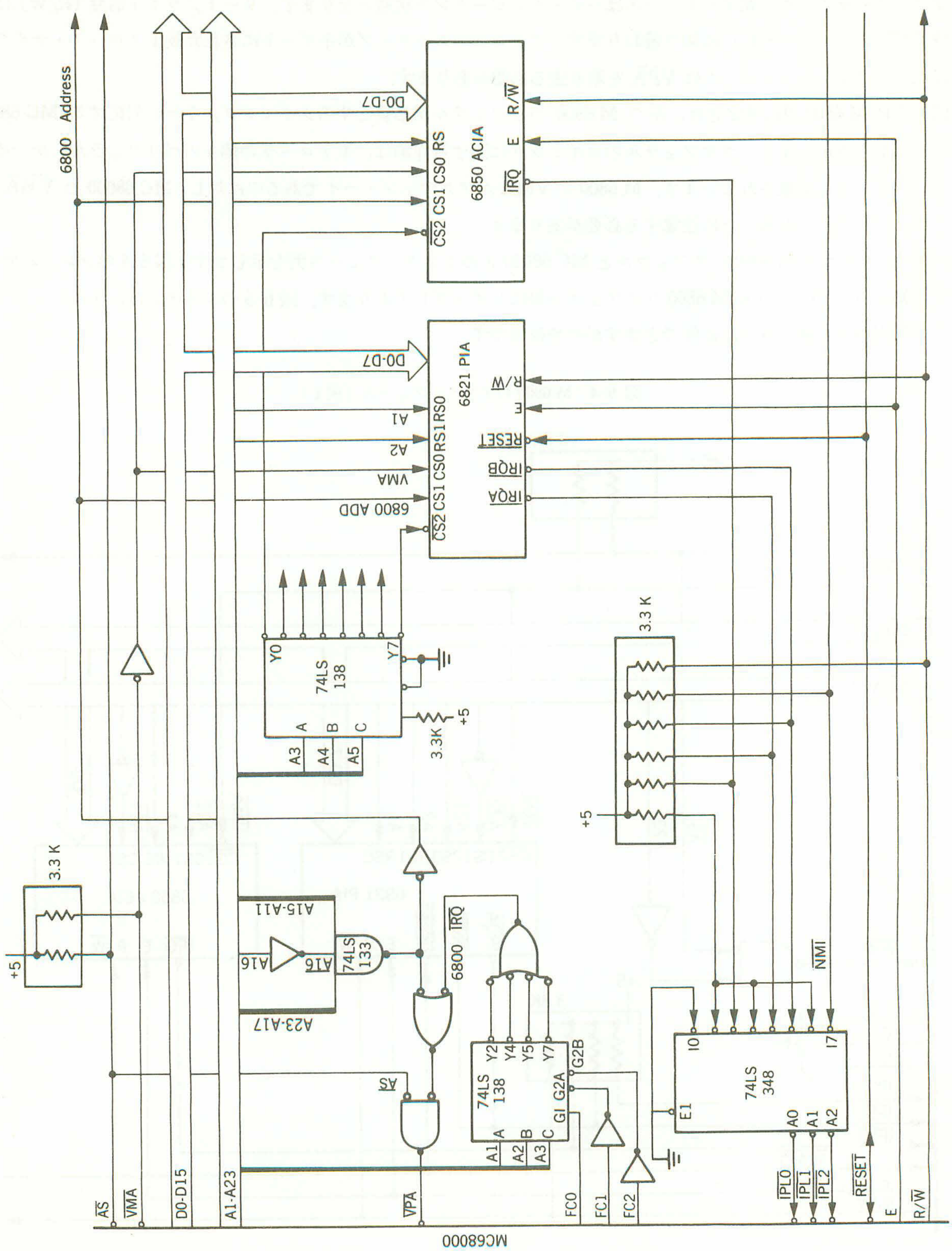


図 6-5　M6800のインターフェース（例2）

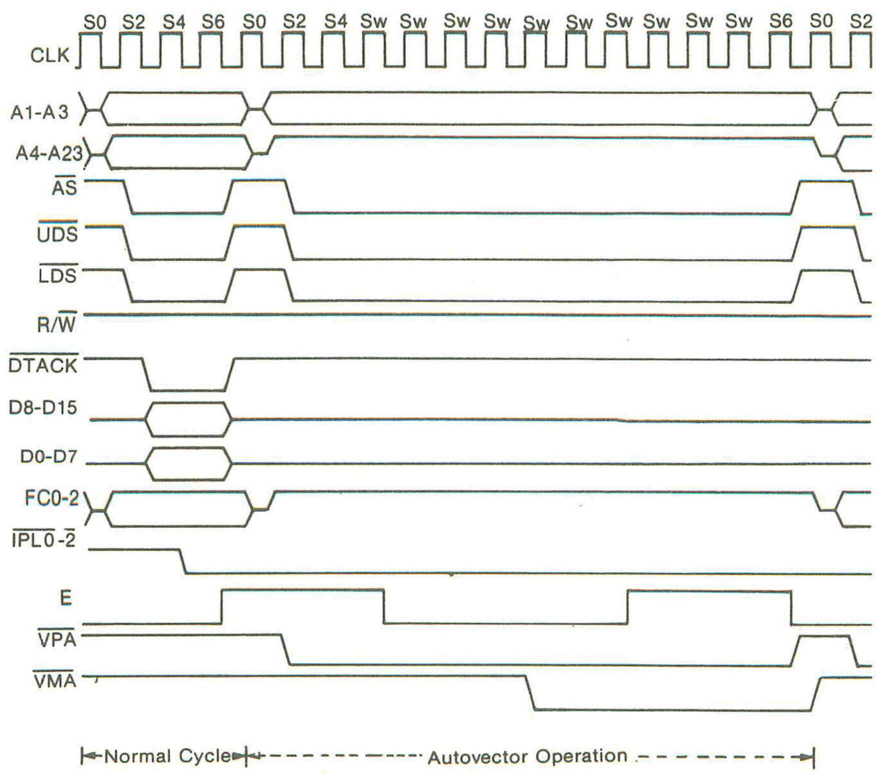
6.2 割込みインターフェース・オペレーション

図6-6に示すように，割込みアクノリッジ・サイクル（Interrupt Acknowledge Cycle）においてプロセッサがベクタをフェッチしているとき $\overline{\text{VPA}}$ を入力しますと，MC 68000は，$\overline{\text{VMA}}$ をアサートにし，正規のM 6800リード・サイクルを完了します．その後，プロセッサは内部で発生したベクタを使って，割込みのサービスを開始します．この処理は，オート・ベクタと呼ばれています．7つのオート・ベクタは，ベクタ番号25から31（10進）に指定されています．

オート・ベクタリングは，M 68000割込みシーケンスと同様に（しかし制限されることなく）動作します．6個のノーマル割込みベクタと1個のNMI型のベクタがあることが基本的な違いです．M6800 と MC 68000 のベクタ方式を使用した割込みは，割込みサービス・ルーチンをアドレス空間のどこのロケーションにでも置けます．これは，ベクタ番号は決められていますが，ベクタ・テーブル・エントリの内容は，ユーザによって決定されるからです．

VMAは，オート・ベクタリング中にアサートになってしまうので，意図しないM 6800ペリフェラルをアクセスしないようにアドレス・デコード回路を入れておかなければなりません．これを2つの例で示します．図6-4の回路例では，MC 68000がベクタをフェッチしている間，アドレスの上位ビット（A 23）が1になっており，これによってオート・ベクタ発生のためのVPAを入力します．しかし，この回路例ではオート・ベクタ割込み方式のみが実行されることに注意してください．図6-5のシステムは，より柔軟な設計になっています．

図 6-6 オート・ベクタ・オペレーションのタイミング図



付録A　コンディション・コード（条件コード）の変化

本付録は，コンディション・コードがどのように設計されたか，コンディション・コードの各ビットがどのような意味をもつか，それらがどのように変化するか，および命令セットがどのように使用されるかについて詳細に述べます．

コンディション・コード設計にあたり，次の二つの点に注意しました．

一貫性をもたせること：（種々の）命令，使用法および事例に対して一貫性をもつこと．

有効な結果をもたせること：有用な情報を提供する時以外は変化しないこと．

命令に対する一貫性とは，命令が一般的な命令セットのある特殊な場合であっても常に同じ方法でコンディション・コードを変化させるということです．事例に対する一貫性とは，もしある命令があるコンディション・コードを変化させたなら，その事例はいつでもそのコンディション・コードを変化させるということです．使用法に対する一貫性とは，コンディション・コードが比較，テスト，あるいは移動命令のどれによりセットされたとしても，コンディション命令は常に同じ状態をテストするということです．コンディション命令に使われるテストとコード変化は，A．4のところで述べます．

A.1　コンディション・コード・レジスタ（CCR：condition code register）

SR（Status Register）の CCR 部は以下の5ビットからなります．

N：負

Z：ゼロ

V：オーバフロー

C：キャリ

X：　拡張

始めの4ビットは，プロセッサの計算結果を示す，本来の意味でのコンディション・コードです．Xビットは多重精度演算のためのオペランドです．プログラミング・モデルを単純化するためにキャリ・ビット（C）と多重精度オペランド用拡張ビット（X）は，MC 68000 では分割されています．

A.2　コンディション・コード・レジスタ（CCR）の表記法

命令セットの詳細説明は付録Bで行いますが，コンディション・コードの変化は，次に示します．

コンディション・コード

X	N	Z	V	C

N（負）　　結果の最上位ビットが1ならセット，0ならクリア．

Z（ゼロ）　　結果が0ならセット，それ以外はクリア．

V（オーバフロー）　　算術オーバフローがあればセット．つまり，結果がオペランド・サイズで表せなければセットされます．そうでなければクリア．

C（キャリ）　　加算でオペランドの最上位ビットからキャリ（桁上げ）がでるとセット．減算でボロー（桁

下げ）が出てもセット．それ以外ならクリア．

X（拡張）　　データ移動には無変化．変化する時はCビットと同じようにセットされます．

コンディション・コード・レジスタ（CCR）を表すのに，便宜上，次の表記法を用います．

＊　オペレーションの結果によりセットされる．

—　オペレーションによって変化しない．

0　クリアされる．

1　セットされる．

U　オペレーションのあと不定となる．

A.3　コンディション・コードの変化

ほとんどのオペレーションは，ソース・オペランド（source operand）とディスティネーション・オペランド（destination operand）を有し，実行し，結果をディスティネーションの位置に格納します．単一オペランド・オペレーションにおいては，ディスティネーション・オペランドのみを有し，計算した結果をディスティネーションの位置に格納します．各命令がどのようにコンディション・コードをセットするかを，表A-1に詳しく示します．

表 A-1 コンディション・コードの計算

Operations	X	N	Z	V	C	Special Definition
ABCD	*	U	?	U	?	C = Decimal Carry $Z = Z \cdot \overline{Rm} \cdot \ldots \cdot \overline{R0}$
ADD, ADDI, ADDQ	*	*	*	?	?	$V = Sm \cdot Dm \cdot \overline{Rm} + \overline{Sm} \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot Dm + \overline{Rm} \cdot Dm + Sm \cdot \overline{Rm}$
ADDX	*	*	?	?	?	$V = Sm \cdot Dm \cdot \overline{Rm} + \overline{Sm} \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot Dm + \overline{Rm} \cdot Dm + Sm \cdot \overline{Rm}$ $Z = Z \cdot \overline{Rm} \cdot \ldots \cdot \overline{R0}$
AND, ANDI, EOR, EORI, MOVEQ, MOVE, OR, ORI, CLR, EXT NOT, TAS, TST	—	*	*	0	0	
CHK	—	*	U	U	U	
SUB, SUBI SUBQ	*	*	*	?	?	$V = \overline{Sm} \cdot Dm \cdot \overline{Rm} + Sm \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot \overline{Dm} + Rm \cdot \overline{Dm} + Sm \cdot Rm$
SUBX	*	*	?	?	?	$V = \overline{Sm} \cdot Dm \cdot \overline{Rm} + Sm \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot \overline{Dm} + Rm \cdot \overline{Dm} + Sm \cdot Rm$ $Z = Z \cdot \overline{Rm} \cdot \ldots \cdot \overline{R0}$
CMP, CMPI, CMPM	—	*	*	?	?	$V = \overline{Sm} \cdot Dm \cdot \overline{Rm} + Sm \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot \overline{Dm} + Rm \cdot \overline{Dm} + Sm \cdot Rm$
DIVS, DIVU	—	*	*	?	0	V = Division Overflow
MULS, MULU	—	*	*	0	0	
SBCD, NBCD	*	U	?	U	?	C = Decimal Borrow $Z = Z \cdot \overline{Rm} \cdot \ldots \cdot \overline{R0}$
NEG	*	*	*	?	?	$V = Dm \cdot Rm$, $C = Dm + Rm$
NEGX	*	*	?	?	?	$V = Dm \cdot Rm$, $C = Dm + Rm$ $Z = Z \cdot \overline{Rm} \cdot \ldots \cdot \overline{R0}$
BTST, BCHG, BSET, BCLR	—	—	?	—	—	$Z = \overline{Dn}$
ASL	*	*	*	?	?	$V = Dm \cdot (\overline{D_{m-1}} + \ldots + \overline{D_{m-r}})$ $+ \overline{Dm} \cdot (D_{m-1} + \ldots + D_{m-r})$ $C = D_{m-r+1}$
ASL (r = 0)	—	*	*	0	0	
LSL, ROXL	*	*	*	0	?	$C = D_{m-r+1}$
LSR (r = 0)	—	*	*	0	0	
ROXL (r = 0)	—	*	*	0	?	C = X
ROL	—	*	*	0	?	$C = D_{m-r+1}$
ROL (r = 0)	—	*	*	0	0	
ASR, LSR, ROXR	*	*	*	0	?	$C = D_{r-1}$
ASR, LSR (r = 0)	—	*	*	0	0	
ROXR (r = 0)	—	*	*	0	?	C = X
ROR	—	*	*	0	?	$C = D_{r-1}$
ROR (r = 0)	—	*	*	0	0	

— Not affected
U Undefined
? Other — see Special Definition

* General Case:
X = C
N = Rm
$Z = \overline{Rm} \cdot \ldots \cdot \overline{R0}$

Sm — Source operand most significant bit
Dm — Destination operand most significant bit
Rm — Result bit most significant bit
n — bit number
r — shift amount

A.4　コンディションのテスト

コンディション付きブランチ命令とセット命令におけるコンディション・コードおよびテストの関係を表 A-2に示します．各コンディションとテストの関係は，コンディション・コードの現在の状態に基づく論理式で示されます．もしこの式の結果が1なら，コンディションが成立し真となり，もし，この式が0ゼロを与えれば，コンディションは不成立で，偽となります．たとえば，Tコンディションは常に成立であり，一方，EQコンディションでは，コンディション・コードのZビットがその時セットされている場合のみ成立します．

表 A-2　コンディションのテスト

Mnemonic	Condition	Encoding	Test
T	true	0000	1
F	false	0001	0
HI	high	0010	$\overline{C} \cdot \overline{Z}$
LS	low or same	0011	$C + Z$
CC	carry clear	0100	$\overline{C}$
CS	carry set	0101	C
NE	not equal	0110	$\overline{Z}$
EQ	equal	0111	Z
VC	overflow clear	1000	$\overline{V}$
VS	overflow set	1001	V
PL	plus	1010	$\overline{N}$
MI	minus	1011	N
GE	greater or equal	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
LT	less than	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
GT	greater than	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
LE	less or equal	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$

付録B 命令セットの詳細

本付録は，MC 68000 命令セットの各命令の詳しい情報をアルファベット順に，ならべています．

B.1 アドレッシング・カテゴリ (Adressing Categories)

実効アドレス・モード（effective address modes）は使われ方により分類できます．以下の分類が命令の定義に使われます．

データ（DATA）：もし実効アドレス・モードがデータ・オペランドを参照する場合，それはデータ・アドレッシング実効アドレス・モードと見なされる．

メモリ（MEMORY）：もし実効アドレス・モードがメモリ・オペランドを参照する場合，それはメモリ・アドレッシング実効アドレス・モードと見なされる．

可変（ALTERABLE）：もし実効アドレス・モードが可変（書き替え可）オペランドを参照する場合，それは可変アドレッシング実効アドレス・モードと見なされる．

制御（CONTROL）：もし実効アドレス・モードが付随のサイズに関係なく，メモリ・オペランドを参照にする場合，それは制御アドレッシング実効アドレス・モードと見なされる．

表 B-1に，それぞれの実効アドレス・モードが属する種々のカテゴリ（categories）を示します．

表B-1 実効アドレッシング・モードのカテゴリ

Effective Address Modes	Mode	Register	Addressing Categories				Assembler Syntax
			Data	Memory	Control	Alterable	
Dn	000	register number	X			X	Dn
An	001	register number				X	An
An@	010	register number	X	X	X	X	(An)
An@+	011	register number	X	X		X	(An)+
An@-	100	register number	X	X		X	-(An)
An@(d)	101	register number	X	X	X	X	d(An)
An@(d, ix)	110	register number	X	X	X	X	d(An, Ri)
xxx.W	111	000	X	X	X	X	xxx
xxx.L	111	001	X	X	X	X	xxxxxx
PC@(d)	111	010	X	X	X		PC relative
PC@(d, ix)	111	011	X	X	X		PC rel. + Ri
#xxx	111	100	X	X			#xxx

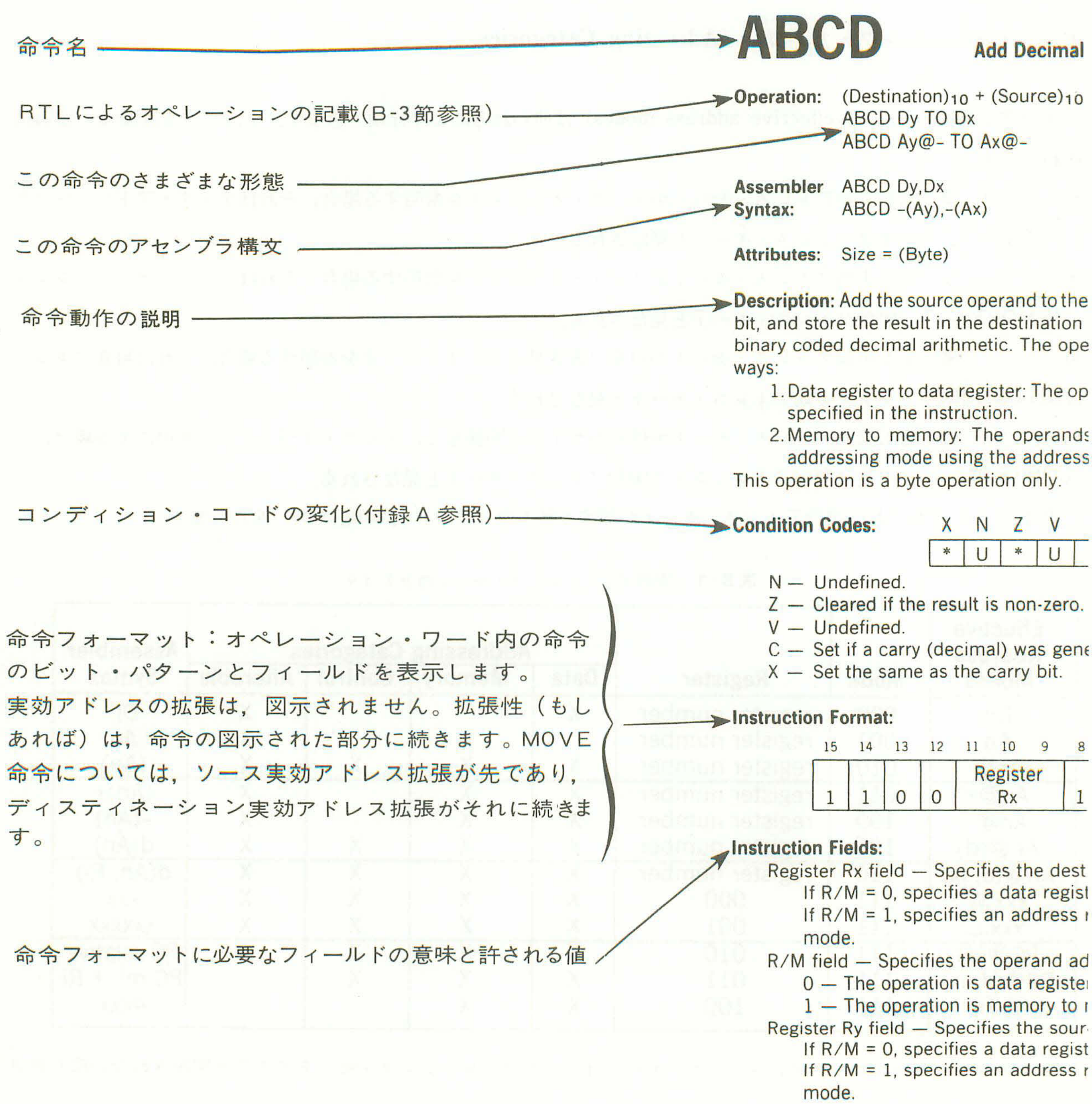
ステータス・レジスタ・アドレッシング・モードは，そのアドレッシング・モードであると明記されない限り許されません．

これらのカテゴリ（Categories）を組み合わせることによって，より詳細な分類が定義できます．たとえば，命令記載では可変メモリまたはデータ可変と分類されて使われていたとすると，前者は可変アドレスとメモリ・アドレスの両方のアドレス指定モードを参照し，後者はデータ・アドレッシング・モードと可変アドレッシング・モードを参照します．

B.2 命　令　説　明

個々の命令フォーマットは以後のページに示されています．図 B-1 にどのような情報が与えられるかを例を使って示します．

図B-1　命　令　の　説　明　例



B.3 RTL (Register Transfer Language：レジスタ転送言語) 定義

以下にレジスタ転送言語定義で命令セットの詳細な動作説明をします．

オペランド：

An：アドレス・レジスタ
Dn：データ・レジスタ
Rn：データ・レジスタまたはアドレス・レジスタのいずれか
PC：プログラム・カウンタ
SR：ステータス・レジスタ
CCR：コンディション・コード (SR の下位バイト)
SSP：スーパバイザ・スタック・ポインタ
USP：ユーザ・スタック・ポインタ
SP：アクティブ・スタック・ポインタ (A7と同じ)

X：拡張オペランド (コンディション・コードからの)
Z：ゼロ・コンディション・コード
V：オーバフロー・コンディション・コード
Immediate Data：イミディエイトデータ・オペランド
d：アドレス変位 (displacement)
Source：ソース位置の指定
Destination：ディスティネーション位置の指定
Vector：エクセプション・ベクタの位置

サブフィールドと修飾記号 (qualifiers)

＜ビット＞OF＜オペランド＞　オペランドのある1つのビットを選ぶ，
＜オペランド＞〔＜ビット番号＞：＜ビット番号＞〕　オペランドのサブフィールドを選ぶ．
(**＜オペランド＞**)　参照されたオペランドの中身
＜オペランド＞$_{10}$　オペランドはBCD (2進化10進) であり，オペレーションは10進で実行される．
＜オペランド＞@＜モード＞　オペランドで指定したアドレス・レジスタがオペランド・データのメモリ位置を指示することを表すレジスタ間接オペレータ．オプションのモード修飾記号は，－，＋，(d)，および (d，ix)．これらは第2章で説明している．

オペレーション：オペレーションは2オペランド操作．単一オペランド操作およびその他に分けられる．

2オペランド操作：これらの命令は＜オペランド＞＜オペレーション＞＜オペランド＞で表わされ，ここでの＜オペレーション＞は次のいずれかである．

→　左のオペランドが右のオペランドで指定される位置に移される
↔　2つのオペランド内容が交換される
＋　両オペランドが加算される
－　右側のオペランドが左側のオペランドから減算される
＊　左右のオペランドが乗算される
／　左のオペランドが右のオペランドで除算される
∧　左右オペランドが論理 AND される
∨　左右オペランドが論理 OR される
⊕　左右オペランドが論理 EOR される
＜　関係テスト，左のオペランドが右のオペランドより小なら真
＞　関係テスト，左のオペランドが右のオペランドより大なら真

≠　関係テスト，左のオペランドが右のオペランドに等しくなければ真

shifted by / rotated by ｝ 左のオペランドが，右のオペランドで規定される位置数だけシフト，またはローテイトされる．

単一オペランド：

～＜**オペランド**＞　オペランドの論理的補数をとる．

＜**オペランド**＞**Sign-extended**　オペランドが符号拡張される．上位半分の全ビットが下位半分の最上位ビットと等しくされる

＜**オペランド**＞**tested**　オペランドが0と比較され，その結果がコンディション・コードのセットに使用する．

その他：

TRAP　PC→SSP@—；SR→SSPⓐ—；(ベクタ)→PCS と等価

STOP　ストップ状態に入り，割込みを待つ

If＜コンディション＞then＜オペレーション＞else＜オペレーション＞

そのコンディションでテストされる．もし真なら then の次の操作が実行さるれ．もしコンディションが偽でオプションの else 部があれば else のの操作が実行されます，もしコンディションが偽でオプションの else 部がなければ，命令は操作されない．

ABCD (Add Decimal with Extend—拡張付き10進加算)

オペレーション：(ディスティネーション)$_{10}$+(ソース)$_{10}$+X→ディスティネーション

ABCD　Dy　TO　Dx

ABCD　Ay@−TO　Ax@−

アセンブラ：ABCD　Dy,　Dx

ABCD−(Ay),−(Ax)

属性：サイズ＝（バイト）

説明：ソース・オペランドを可拡張ビット(X)とともにディスティネーション・オペランドに加え，結果をディスティネーション位置にストアします．加算はBCD（2進化10進）演算を用いて実行されます．オペランドは次の2つの方法のアドレスができます．

1. データ・レジスタからデータ・レジスタ：両オペランドは命令で指定されるデータ・レジスタ内にある．
2. メモリからメモリ：両オペランドは命令び指定されるアドレス・レジスタを用いたプリディクリメント(predecrement)アドレッシング・モードでアドレスされる．

この命令は，バイト・オペレーションのみです．

コンディション・コード：

X	N	Z	V	C
*	U	*	U	*

N——不定

Z——結果が0でなければクリア，それ以外の時は変化しない．

V——不定

C——キャリ（10進）が発生したらセット，それ以外の時はクリア．

X——キャリ・ビットと同じ．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	レジスタ Rx			1	0	0	0	0	R/M	レジスタ Ry		

命令フィールド：

レジスタ　Rx　フィールド——ディスティネーションのレジスタ番号を指定

R/M=0，データ・レジスタを指定．

R/M=1，プリデクリメント・アドレス・モードで使うアドレス・レジスタを指定．

R/M　フィールド——オペランドのアドレッシング・モードを指定．

0——データ・レジスタからデータ・レジスタへのオペレーション．

1——メモリからメモリへのオペレーション.

レジスタ　Ry　フィールド——ソースのレジスタ番号を指定.

R/M=0, データ・レジスタの番号を指定.

R/M=1, プリデクリメント・アドレス・モードで使うアドレス・レジスタの番号を指定.

MC68,000 — ADD BCD STRINGS

```
*      ADD BCD STRINGS
*
*UPON ENTERING THIS ROUTINE, A0 IS
*POINTING TO THE END+1 OF THE FIRST
*STRING, AND A1, THE SECOND. THE NUM-
*BER OF CHARACTERS -1 TO BE ADDED IS IN D1.
*
 ORG $2000
 MOVE #0, CCR
AGAIN ABCD -(A0), -(A1)
 DBRA D1, AGAIN
 RTS
 END
```

ADD (Add Binary 2進加算)

オペレーション：(ディスティネーション)+(ソース)→ディスティネーション
ADD<ea>TO Dn
ADD Dn TO<ea>

アセンブラ：ADD<ea>, Dn
ADD Dn, <ea>

属性：サイズ＝（バイト，ワード，ロングワード）

説明：ソース・オペランドをディスティネーション・オペランドに加え，結果をディスティネーションの位置にストアします．オペレーションのサイズは，バイト，ワード，ロングワードのいずれにも指定できます．命令のモードで，どのオペランドがソースで，どのオペランドがディスティネーションか，およびオペランド・サイズを指定します．

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

N――結果が負であればセット，それ以外の時はクリア．
Z――結果が0であればセット，それ以外の時はクリア．
V――オーバフローが発生すればセット，それ以外の時はクリア．
C――キャリィが発生すればセット，それ以外の時はクリア．
X――キャリィ・ビットと同じ．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	レジスタ			Op-モード			実効アドレス					

命令フィールド：

レジスタ・フィールド――データ・レジスタの番号を指定する．

Op（オペレーション）-モード・フィールド――

バイト	ワード	ロングワード	オペレーション
000	001	010	(<Dn>)+(<ea>)→<Dn>
100	101	110	(<ea>)+(<Dn>)→<ea>

実効アドレス・フィールド――アドレッシング・モードを指定．

a．もし指定された位置がソース・オペランドなら，すべてのアドレッシング・モードが可．もし，オペレーション・サイズがバイトなら，アドレス・レジスタ直接アドレッシング・モードは不可．

b．もし指定された位置がディスティネーション・オペランドなら，アドレッシング・カテゴリのメモリと可変のアドレッシング・モードのみ可．

ADDA（Add Address—アドレス加算）

オペレーション：（ディスティネーション）＋（ソース）→ディスティネーション

ADD<ea>TO An

アセンブラ：ADD<ea>, An

属性：サイズ＝（ワード，ロングワード）

説明：ソース・オペランドをディスティネーションのアドレス・レジスタに加え，結果をアドレス・レジスタに格納します．オペレーションのサイズはワードかロングワードに指定できます．

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化せず．

Z——変化せず．

V——変化せず．

C——変化せず．

X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	レジスタ			Op-モード			実効アドレス					

命令フィールド：

レジスタ・フィールド——アドレス・レジスタの番号（常にディスティネーション）を指定．

Op-モード・フィールド——オペレーション・サイズを指定．

011——ワード・オペレーション．ソース・オペランドがロングワードに符号拡張され，アドレス・レジスタ全32ビットがオペレーションされる．

111——ロングワード・オペレーション．

実効アドレス・フィールド——ソース・オペランドのアドレッシング・モードを指定．すべてのアドレッシング・モードが可．

ADDI (Add Immediate イミディエイト加算)

オペレーション：(ディスティネーション)＋イミディエイト・データ→ディスティネーション
ADD #<data>TO<ea>

アセンブラ：ADDI #<data>, <ea>

属性：サイズ＝(バイト，ワード，ロングワード)

説明：イミディエイト・データをディスティネーションの内容に加算し，その結果をディスティネーションの位置に格納します．オペレーションのサイズは，バイト，ワード，ロングワードに指定できます．イミディエイト・データのサイズはオペレーション・サイズに合致できます．

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

N――結果が負があればセット，それ以外の時はクリア．
Z――結果が0であればセット，それ以外の時はクリア．
V――オーバフローが発生すればセット，それ以外の時はクリア．
C――キャリが発生すればセット，それ以外の時はクリア．
X――キャリ・ビットと同じ．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	サイズ		実効アドレス					
ワード・データ(16ビット)								バイト・データ(8ビット)							
ロング・データ(32ビット、先行ワードを含む)															

命令フィールド：

サイズ・フィールド――オペレーション・サイズを指定する．
00――バイト・オペレーション．
01――ワード・オペレーション．
10――ロングワード・オペレーション．

実効アドレス・フィールド――ディスティネーション・オペランドのアドレッシング・モードを指定する．データと可変のアドレッシング・モードのみ可

イミディエイト・フィールド――(命令の直後のデータが入る)
サイズ＝00，データはイミディエイト・ワードの下位ビイト(8ビット)
サイズ＝01，データはイミディエイト・ワード(16ビット)
サイズ＝10，データは次の2つのイミディエイト・ワード(32ビット)

ADDQ (Add Quick 即加算)

オペレーション：(ディスティネーション)+イミディエイト・データ→ディスティネーション,

ADDQ #<data>TO<ea>

アセンブラ：ADDQ #<data>, <ea>

属性：サイズ＝(バイト，ワード，ロングワード)

説明：イミディエイト・データをディスティネーションの位置のオペランドに加え，その結果をディスティネーションの位置に格納します．データの範囲は 1—8．オペレーション・サイズはバイト，ワード，ロングワードに指定できます．アドレス・レジスタへのワード，ロングワード・オペレーションも可能です．

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

N——結果が負であればセット，それ以外の時はクリアする．

Z——結果が0であればセット，それ以外の時はクリア．

V——オーバフローが発生すればセット，それ以外の時はクリア．

C——キャリが発生すればセット，それ以外の時はクリア．

X——キャリ・ビットと同じ．

アドレス・レジスタへの加算の場合はコンディション・コードは変化しない．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	データ			0	サイズ		実効アドレス					

命令フィールド：

データ・フィールド——3ビットのイミディエイト・データであり，0は8を，1—7は1から7を表わしている．

サイズ・フィールド——オペレーション・サイズを指定する．

00——バイト・オペレーション．

01——ワード・オペレーション．

10——ロングワード・オペレーション．

実効アドレス・フィールド——ディスティネーション・オペランドのアドレッシング・モードを指定する．可変アドレッシング・モードのみ可．ただし，オペレーション・サイズがバイトなら，アドレス・レジスタ直接のアドレッシング・モードは不可．

ADDX (Add Extendid 拡張加算)

オペレーション：(ディスティネーション)+(ソース)+X→ディスティネーション
ADDX　Dy　TO　Dx
ADDX　Ay@−TO　Ax@−

アセンブラ：ADDX　Dy, Dx
ADDX−(Ay), −(Ax)

属性：サイズ＝（バイト，ワード，ロングワード）

説明：ソース・オペランドと拡張ビット(X)を行き先オペランドに加え，結果をディスティネーションの位置に格納します．オペランドは2つの方法でアドレスできます．

1. データ・レジスタからデータ・レジスタ：オペランドは命令で指定されるデータ・レジスタ内にある．
2. メモリからメモリ：オペランドは命令で指定されるアドレス・レジスタを用いたプリデクリメント・アドレッシング・モードでアドレスされる．

オペレーションのサイズはバイト，ワード，ロングワードに指定できます．

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

N——結果が負であればセット，それ以外の時はクリア．
Z——結果が0であればセット，それ以外の時はクリア．
V——オーバーフローが発生すればセット，それ以外の時はクリア．
C——キャリが発生すればセット，それ以外の時はクリア．
X——キャリ・ビットと同じ．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	レジスタ Rx			1	サイズ		0	0	R/M	レジスタ Ry		

命令フィールド：

レジスタ　Rx　フィールド——ディスティネーション・レジスタの番号を指定
R/M=0，データ・レジスタの番号を指定．
R/M=1，プリデクリメント・アドレス・モードで使うアドレス・レジスタの番号を指定．

サイズ・フィールド——オペレーション・サイズを指定．
00——バイト・オペレーション．
01——ワード・オペレーション．

10——ロングワード・オペレーション.

R/M フィールド——オペランドのアドレッシング・モードを指定.

0——データ・レジスタからデータ・レジスタ.

1——メモリからメモリ.

レジスタ Ry フィールド——ソース・レジスタの番号を指定.

R/M=0, データ・レジスタの番号を指定.

R/M=1, プリデクリメント・アドレス・モードで使うアドレス・レジスタの番号を指定.

AND (AND Logical 論理 AND)

オペレーション：(ディスティネーション)∧(ソース)→ディスティネーション

アセンブラ：AND<ea>, Dn

AND Dn,<ea>

属性：サイズ＝（バイト，ワード，ロングワード）

説明：ソース・オペランドをディスティネーション・オペランドにANDし，結果をディスティネーションの位置にストアします．オペレーション・サイズはバイト，ワード，ロングワードを指定できます．アドレス・レジスタの内容はオペランドとして使えません．

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

N——結果の最上位ビットが1であればセット，それ以外の時，クリア．

Z——結果が0であればセット，それ以外の時，クリア．

V——常にクリア．

C——常にクリア．

X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	レジスタ			Op-モード			実効アドレス					

命令フィールド：

レジスタ・フィールド——データ・レジスタの番号を指定．

Op-モード・フィールド——

バイト	ワード	ロングワード	オペレーショソ
000	001	010	(<Dn>)∧(<ea>)→<Dn>
100	101	110	(<ea>)∧(<Dn>)→<ea>

実効アドレス・フィールド——アドレッシング・モードを指定．

ソース・オペランドが指定された場合は，データ・アドレッシング・モードのみ可．

ディスティネーション・オペランドが指定された場合は，可変とメモリのアドレッシング・モードのみ可．

ANDI (AND Immediate　イミディエイトAND)

オペレーション：(ディスティネーション)∧イミディエイト・データ→ディスティネーション
AND　#＜data＞TO＜ea＞

アセンブラ：ANDI　#＜data＞, ＜ea＞

属性：サイズ＝（バイト，ワード，ロングワード）

説明：イミディエイト・データをソース・オペランドにANDし，結果をディスティネーションの位置にストアします．オペレーション・サイズは，バイト，ワード，ロングワードを指定できます．イミディエイト・データのサイズはオペレーション・サイズに合致します．ステータス・レジスタへのANDは2つのモードで可能です．

1. オペレーション・サイズがバイトなら，オペレーションはコンディション・コードを含むSRの下位バイトのみに対し実行される．
2. オペレーショソ・サイズがワードなら，ステータス・レジスタ全体に対し実行される・これは特権操作である．

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

N——結果の最上位ビットが1であかばセット，それ以外の時，クリア．
Z——結果が0であればセット，それ以外の時，クリア．
V——常にクリア．
C——常にクリア．
X——変化せず．
ステータス・レジスタをアクセスするとコンディション・コードはオペレーションにより変化する．

命令フォーマット：

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td colspan="2">サイズ</td><td colspan="6">実効アドレス</td></tr>
<tr><td colspan="8">ワード・データ(16ビット)</td><td colspan="8">バイト・データ(8ビット)</td></tr>
<tr><td colspan="16">ロング・データ(32ビット、先行ワードを含む)</td></tr>
</table>

命令フィールド

サイズ・フィールド——オペレーション・サイズを指定．
00——バイト・オペレーション．
01——ワード・オペレーション．
10——ロングワード・オペレーション．

実効アドレス・フィールド——ディスティネーションのオペランドを指定．データと可変のアドレッシング・モードおよびステータス・レジスタへのアドレッシング・モードのみ可．

イミディエイト・フィールド——（命令の直後のデータ）

サイズ＝00，データはイミディエイト・ワードの下位バイト（8ビット）．

サイズ＝01，データはイミディエイト・ワード（16ビット）．

サイズ＝10，データは次の2つのイミディエイト・ワード（32ビット）．

ASL, ASR (Arithmetic Shift　算術シフト)

オペレーション：(ディスティネーション) Shifted by<カウント>→ディスティネーション

ASd　Dy　BY　Dx

ASd　Dy　BY　#<data>

ASd<ea>

(注)　dは方向を示しRまたはL.

アセンブラ：ASd　Dx, Dy

ASd　#<data>, Dy

ASd<ea>

属性：サイズ=<バイト，ワード，ロングワード>

説明：ディスティネーション・オペランドのビットを指定方向に算術シフトします．キャリおよびXは，オペランドから最後に送り出されたビットを受け取ります．レジスタをシフトする場合のシスト回数(shift range)は2つの方法で指定できます．

1. イミディエイト：シフト回数は命令で指定される（1～8回)．
2. レジスタ：シフト回数は命令で指定されるデータ・レジスタにある．

オペレーション・サイズは，バイト，ワード，ロングワードが使用できます．ただし，メモリ内容の場合は1ビットしかシフトできず，オペレーション・サイズもワードのみに限定されます．

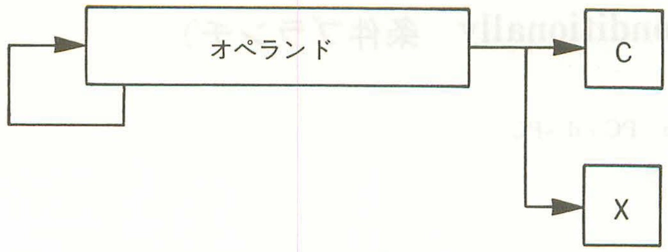
ASL ではオペランドは左シフトされ，シフトされた位置数がシフト回数です．最上位が送り出されたビットはキャリと拡張のビットに入ります．最下位ビットは0が入ります．シフト中に一度でも符号が変化すると，オーバフロー・ビットが示されます．

ASL



ASR ではオペランドは右シフトされる．シフトされた位置数がシフト回数．最下位から送り出されたビットがキャリと拡張の両ビットに入る．符号ビットは最上位ビットに再度入れられる．

ASR



コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

N――結果の最上位ビットが1であればセット，それ以外の時はクリア．

Z――結果が0であればセット，それ以外の時はクリア．

V――シフト操作中に最上位ビットが一回でも変化すればセット，それ以外の時はクリア．

C――オペランドから最後に送り出されたビットに従ってセット．シフト回数が0の時クリア．

X――オペランドから最後に送り出されたビットに従ってセット，シフト回数が0の時は変化せず．

命令フォーマット（レジスタ・シフト）：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	カウント/レジスタ			dr	サイズ		i/r	0	0	レジスタ		

命令フィールド（メモリ・シフト）：

dr　フィールド――シフト方向を指定．

0――右シフト

1――左シフト

実効アドレス・フィールド――シフトされるオペランドを指定．メモリと可変のアドレッシング・モードのみ可．

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	0	dr	1	1	実効アドレス					

Bcc (Branch Conditionally　条件ブランチ)

オペレーション： If　cc　then　PC＋d→PC
Bcc＜ラベル＞

アセンブラ： Bcc＜ラベル＞

属性： サイズ＝（バイト，ワード）

説明： 指定されたコンディションが満足されると，プログラムの実行はプログラム・カウンタ（PC）＋ディスプレースメントにブランチします．ディスプレースメントは“2”の補数の整数で，相対距離をバイトで表します．PC 内の値は現在の命令の位置＋2です．もし命令語内の8ビットディスプレースメントが0なら，16ビットのディスプレースメント（命令の直後のワード）が使われます．“cc”として，以下の条件を指定できます．

CC	carry clear	LS	low or same
CS	carry set	LT	less
EQ	equal	MI	minus
GE	greater or equal	NE	not equal
GT	greater	PL	plus
HI	high	VC	no overflow
LE	less or equal	VS	overflow

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化せず.
Z——変化せず.
V——変化せず.
C——変化せず.
X——変化せず.

命令フォーマット：

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td colspan="4">条件</td><td colspan="8">8ビット変位</td></tr>
<tr><td colspan="16">8ビット変位＝0の時、16ビット変位</td></tr>
</table>

命令フィールド

コンディション・フィールド——次の14のコードのいずれかを指定する.

CC	0100	LS	0011

CS —— 0101	LT —— 1101
EQ —— 0111	MI —— 1011
GE —— 1100	NE —— 0110
GT —— 1110	PL —— 1010
HI —— 0010	VC —— 1000
LE —— 1111	VS —— 1001

8ビット　ディスプレースメント・フィールド——コンディションが満足された場合，ブランチ命令と次に実行すべき命令との相対距離（バイト単位）を示す“2”の補数の整数.

16ビット　ディスプレースメント・フィールド——8ビットより大きな相対距離を示す場合に使う（8ビットディスプレースメントはこのとき0である).

BCHG（Test a Bit and Change　ビット・テストと変更）

オペレーション：～(＜ビット番号＞　OF　ディスティネーション→Z；
～(＜ビット番号＞　OF　ディスティネーション)→＜ビット番号＞　OF　ディスティネーション
BCHG　Dn　OF＜ea＞
BCHG　♯＜データ＞OF＜ea＞

アセンブラ：BCHG　Dn，＜ea＞

BCHG　♯＜データ＞，＜ea＞

属性：サイズ＝（バイト，ロングワード）

説明：ディスティネーション・オペランドの内，ソース・オペランドで示されるビットがテストされ，そのビットの状態がコンディション・コードZに反映されます．テストの後，そのビットの状態はディスティネーションで反転されます．もしデータ・レジスタがディスティネーションならビット番号はデータ・レジスタの全ビットのオペレーションを可能にするため，モジュロ-32（つまりビット0から31まで許される）となります．もしメモリ位置がディスティネーションなら，その位置から1バイトが読込まれ，ビット番号モジュロ-8（つまりビット0～7）を使ってそのビットがオペレーションされ，1バイトがもとの位置に書き戻されます．このビット番号は2つの方法で指定できます．

1. イミディエイト：ビット番号は命令の第2ワードで指定される．
2. レジスタ：ビット番号は命令で指定されるレジスタ内にある．

コンディション・コード：

X	N	Z	V	C
—	—	*	—	—

N――変化せず．
Z――テストされたビットが0であればセット，それ以外の時はクリア．
V――変化せず．
C――変化せず．
X――変化せず．

命令・フォーマット（ビット番号ダイナミック）：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	レジスタ			1	0	1	実効アドレス					

命令フィールド（ビット番号ダイナミック）：

レジスタ・フィールド――ビット番号を格納しているデータ・レジスタの番号を指定．
実効アドレス・フィールド――ディスティネーション・オペランドを指定．データと可変のアドレッシング・モードのみ可．

命令フォーマット（ビット番号スタティック）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	1	実効アドレス					
0	0	0	0	0	0	0	0	ビット番号							

命令フィールド（ビット番号スタティック）：

実効アドレス・フィールド――ディスティネーション・オペランドを指定．データと可変のアドレッシング・モードのみ可．

ビット番号フィールド――ビット番号を指定．

BCLR（Test a Bit and Clear　ビット・テストとクリア）

オペレーション：～(＜ビット番号＞　OF　ディスティネーション)→Z
0→＜ビット番号＞→OF　ディスティネーション
BCLR　Dn　OF　＜ea＞
BCLR　#＜データ＞，OF＜ea＞

アセンブラ：BCLR　Dn，＜ea＞
BCLR　#＜データ＞，＜ea＞

属性：サイズ＝（バイト，ロングワード）

説明：ディスティネーション・オペランドの内，ソース・オペランドで示される1ビットがテストされ，そのビットの状態がコンディション・コードZに反映されます．テストの後，そのディスティネーションでクリアされます．もしデータ・レジスタがディスティネーションなら，ビット番号は，データ・レジスタの全ビットのオペレーションを可能にするためにモジュロ-32（ビット0から31まで許される）となります．もし，メモリ位置がディスティネーションなら，その位置から1バイトが読み込まれ，ビット番号モジュロ-8（ビット0から7まで）を使ってそのビットがオペレーションされ，1バイトがもとの位置に書き戻されます．このビット番号は，2つの方法で指定できます．

1. イミディエイト：ビット番号は命令の第2ワードで指定される．
2. レジスタ：ビット番号は命令で指定されるデータ・レジスタ内にある．

コンディション・コード：

X	N	Z	V	C
—	—	*	—	—

N——変化せず．
Z——テストされたビットが0の場合セット．それ以外の時クリア．
V——変化せず．
C——変化せず．
X——変化せず．

命令フォーマット（ビット番号ダイナミック）：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	レジスタ			1	1	0	実効アドレス					

命令フィールド（ビット番号ダイナミック）：

レジスタ・フィールド——ビット番号を格納しているデータ・レジスタの番号を指定．
実効アドレス・フィールド——ディスティネーション・オペランドを指定．データと可変アドレッシング・モードのみ可．

命令フォーマット（ビット番号スタティック）：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	0	実効アドレス					
ビット番号															

命令フィールド（ビット番号スタティック）：

実効アドレス・フィールド——ディスティネーション・オペランドを指定．データと可変のアドレッシング・モードのみ可．

ビット・ナンバ・フィールド——ビット番号を指定．

BRA (Branch Always 無条件ブランチ)

オペレーション： PC＋d→PC

BRA<ラベル>

アセンブラ： BRA<ラベル>

属性： サイズ＝（バイト，ワード）

説明： プログラム実行がプログラム・カウンタ（PC)＋ディスプレースメント(d)の位置から続行されます．ディスプレースメントは“2”の補数の整数で，相対距離をバイトで表したものです．PC 内の値は現在の（命令の）位置＋2です．もし命令語内の8ビット変位が0なら，16ビット・ディスプレースメント(命令直後のワード)が使われます．

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化せず.
Z——変化せず.
V——変化せず.
C——変化せず.
X——変化せず.

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	8ビット変位							
8ビット変位＝0の時、16ビット変位															

命令フィールド：

8ビット変位・フィールド——このブランチ命令と次に実行すべき命令との相対距離（バイト単位）を示す“2”の補数の整数.

16ビット・ディスプレースメント・フィールド——8ビットより大きな相対距離を示す場合に使用（8ビット変位はこの時0です）.

BSET (Test a Bit and Set ビット・テストとセット)

オペレーション：～(<ビット番号) OF ディスティネーション→Z

1→<ビット番号>OF ディスティネーション

BSET Dn, OF <ea>

BSET ♯<データ>OF<ea>

アセンブラ：BSET Dn <ea>

BSET ♯<データ>, <ea>

属性：サイズ＝（ワード，ロングワード）

説明：ディスティネーション・オペランドの内，ソース・オペランドで示されるビットがテストされ，そのビットの状態がコンディション・コードに反映されます．テスト後，そのビットはディスティネーションでセットされます．もしデータ・レジスタがディスティネーションなら，ビット番号はデータ・レジスタの全ビットの操作を可能にするため，モジュロ-32（ビット0～31）となります．もしメモリ位置がディスティネーションなら，その位置から1バイトが読込まれ，ビット番号モジュロ-8（ビット0～7）を用いてそのビットがオペレーションされ，1バイトが元の位置に書き戻されます．このビット番号は2つの方法で指定できます．

1. イミディエイト：ビット番号は命令の第2ワードで指定されます．
2. レジスタ：ビット番号は命令で指定されるレジスタ内にあります．

コンディション・コード：

X	N	Z	V	C
—	—	*	—	—

N――変化せず．

Z――テストされたビットが0の場合セット，それ以外の時クリア．

V――変化せず．

C――変化せず．

X――変化せず．

命令フォーマット（ビット番号スタティック）：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	1	実効アドレス					
ビット番号															

命令フィールド（ビット番号スタティック）：

実効アドレス・フィールド――ディスティネーション・オペランドを指定，データと可変のアレッシング・モードのみ可．

ビット番号フィールド――ビット番号を指定．

BSR (Branch to Subroutine サブルーチン・ブランチ)

オペレーション：PC→SP@－；PC＋d→PC
BSR＜ラベル＞

アセンブラ：BSR＜ラベル＞

属性：サイズ＝（バイト，ワード）

説明：BSR 命令の直後の命令のアドレスがシステム・スタック(System Stack)に格納されます．その後プログラム実行はプログラム・カウンタ（PC)＋ディスプレースメントの位置から続行されます．ディスプレースメントは“2”の補数の整数で，相対距離をバイトで表したものです．PC 内の値は現在の（命令の）位置＋2です．命令語内の8ビット・ディスプレースメントがもし0ならば，16ビットのディスプレースメント（命令の直後ワード）が使われます．

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化せず．
Z——変化せず．
V——変化せず．
C——変化せず．
X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	8ビット変位							
8ビット変位＝0の時、16ビット変位															

命令フィールド：

8ビット・ディスプレースメント・フィールド——このブランチ命令と次に実行すべき命令の相対距離（バイト単位）を示す“2”の補数の整数．

16ビット・ディスプレースメント・フィールド——8ビットより大きな相対距離を示す場合に使用（8ビット・ディスプレースメントはこの時0）

BTST (Test a Bit (ビット・テスト)

オペレーション：~(<ビット番号>OF　ディスティネーション)→Z

BTST　Dn　OF<ea>

BTST　♯<データ>OF<ea>

アセンブラ：BTST　Dn, <ea>

BTST　♯<データ>, <ea>

属性：サイズ＝（バイト，ロングバイト）

説明：ディスティネーション・オペランドの内ソース・オペランドで示されるビットがテストされ，そのビットの状態がコンディション・コードZに反映されます．もしデータ・レジスタがディスティネーションなら，ビット番号はデータ・レジスタの 全ビットのオペレンションを可能にするため モジュロ-32（ビット0～31）となります．もしメモリ位置がディスティネーションならその位置から1バイトが読み込まれ，ビット番号モジュロ-8（ビット0～7）を用いてそのビットが操作されます．このビット番号は2つの方法で指定できます．

1. イミディエイト：ビット番号は命令の第2ワードで指定される．
2. レジスタ：ビット番号は命令で指定されるデータ・レジスタ内にある．

コンディション・コード：

X	N	Z	V	C
—	—	*	—	—

N――変化せず．

Z――テストされたビットが0の場合セット，それ以外の時はクリア．

V――変化せず．

C――変化せず．

X――変化せず．

命令フォーマット（ビット番号ダイナミック）：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	レジスタ			1	0	0	実効アドレス					

命令フィールド（ビット番号ダイナミック）：

レジスタ・フィールド――ビット番号を格納しているデータ・レジスタの番号を指定．

実効アドレス・フィールド――ディスティネーション・オペランドを指定．アドレッシング・カテゴリのデータと可変のアドレッシング・モードのみ可．

命令フォーマット（ビット番号スタティック）：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	0	実効アドレス					
ビット番号															

命令フィールド（ビット番号スタティック）：

実効アドレス・フィールド——ディスティネーション・オペランドを指定. イミディエイト・アドレッシング・モードを除いたアドレッシング・カテゴリのデータと可変のアドレッシング・モードのみ可.

ビット番号フィールド——ビット番号を指定.

CHK (Check Register against Bounds　レジスタ限界チェック)

オペレーション：if　Dn<O or Dn> (<ea>) then TRAP
CHK　Dn　WITH　<ea>

アセンブラ：CHK<ea>, Dn

属性：サイズ＝（ワード）

説明：命令で指定されるデータ・レジスタ内の下位ワードが調べられ，ソース・オペランドで示される上限と比較されます．上限は“2”の補数の整数です．もしレジスタ値が0未満またはオペランド・ワード内の上限値より大であると，プロセッサはエクセプション・プロセッシングを開始し，CHK 命令のエクセプション・ベクタを参照するためにベクタ番号が生成されます．

コンディション・コード：

X	N	Z	V	C
—	*	U	U	U

N——Dn<0 であればセット．Dn>(<ea>) であればクリア．それ以外の時は不定．
Z——不定．
V——不定．
C——不定．
X——変化せず．

命令フォーマット：

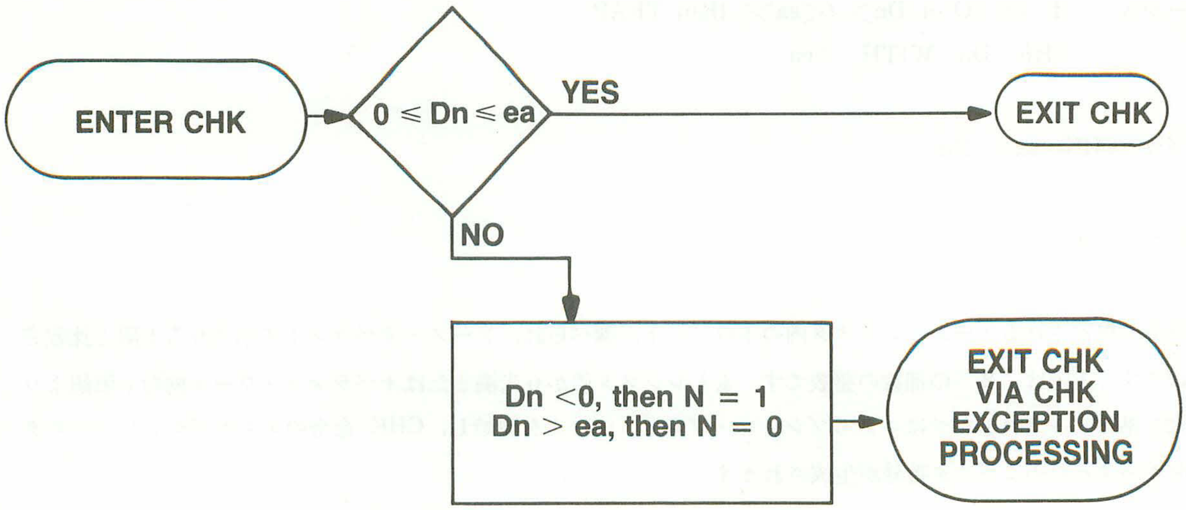
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	レジスタ			1	1	0	実効アドレス					

命令フィールド：

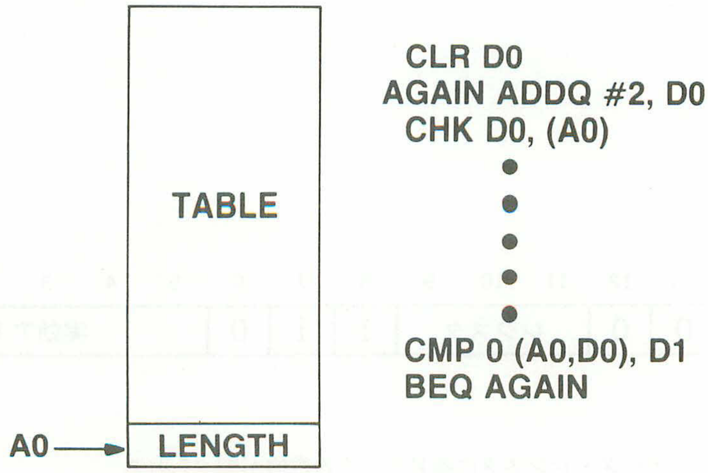
レジスタ・フィールド——データ・レジスタの番号で，この内容が調べられる．

実効アドレス・フィールド——ソース・オペランドを指定して，この内容が上限値．データ・アドレッシング・モードのみ可．

MC68,000 — CHK INSTRUCTION



MC68,000 — CHK INSTRUCTION EXAMPLE



CLR (Clear an Operand　クリア)

オペレーション： 0 ⟶ディスティネーション

CLR <ea>

アセンブラ： CLR <ea>

属性： サイズ＝(バイト，ワード，ロングワード)

説明： ディスティネーションの全ビットが0にクリアされます．オペレーション・サイズはバイト，ワード，ロングワードに指定できます．

コンディション・コード：

X	N	Z	V	C
—	0	1	0	0

N——常にクリア．
Z——常にセット．
V——常にクリア．
C——常にクリア．
X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	サイズ		実効アドレス					

命令フィールド：

サイズ・フィールド——オペレーション・サイズを指定．

00——バイト・オペレーション
01——ワード・オペレーション
10——ロングワード・オペレーション

実効アドレス・フィールド——ディスティネーション・オペランドを指定．データと可変のアドレッシング・モードのみ可．

CMP (Compare 比較)

オペレーション：(ディスティネーション)－(ソース)
CMP <ea>, TO Dn

アセンブラ：CMP <ea>, Dn

属性：サイズ＝(バイト, ワード, ロングワード)

説明：ソース・オペランドをディスティネーション・オペランドから減算し，その結果によりコンディション・コードをセットします．ディスティネーションの位置の内容は変わりません．オペレーションのサイズはバイト，ワード，ロングワードに指定できます．

コンディション・コード：

X	N	Z	V	C
—	*	*	*	*

N——結果が負であればセット，それ以外の時はクリア．
Z——結果が0であればセット，それ以外の時はクリア．
V——オーバフローが発生すればセット，それ以外の時はクリア．
C——ボローが発生すればセット，それ以外の時はクリア．
X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	レジスタ			Op-モード			実効アドレス					

命令フィールド：

レジスタ・フィールド——ディスティネーションのデータ・レジスタ番号を指定．
Op-モード・フィールド——

バイト	ワード	ロングワード	オペレーション
000	001	010	(<Dn>)－(<ea>)

実効アドレッシング・フィールド——ソース・オペランドを指定，すべてのアドレッシング・モードが可．ただし，オペレーション・サイズがバイトであれば直接アドレス・レジスタのアドレッシング・モードは不可．

CMPA (Compare Address アドレス比較)

オペレーション：(ディスティネーション)−(ソース)
CMP <ea>, TO An

アセンブラ：CMPA <ea>, An

属性：サイズ＝(ワード，ロングワード)

説明：ソース・オペランドをディスティネーション・アドレス・レジスタから減算し，その結果に応じてコンディション・コードをセットします．アドレス・レジスタの内容は変わりません．オペレーションのサイズはワードかロングワードとして指定できます．

コンディション・コード：

X	N	Z	V	C
—	*	*	*	*

N——結果が負であればセット，それ以外の時はクリア．
Z——結果が0であればセット，それ以外の時はクリア．
V——オーバフローが発生すればセット，それ以外の時はクリア．
C——ボローが発生すればセット，それ以外の時はクリア．
X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	レジスタ			Op-モード			実効アドレス					

命令フィールド：

レジスタ・フィールド——ディスティネーションのアドレス・レジスタ番号．

Op-モード・フィールド——オペレーション・サイズを指定．

011——ワード・オペレーション．ソース・オペランドは符号拡張され，アドレス・レジスタの全32ビットを使用．

111——ロングワード・オペレーション．

実効アドレス・フィールド——ソース・オペランドを指定．すべてのアドレッシング・モードが可．

CMPI (Compare Immediate イミディエイト比較)

オペレーション：(ディスティネーション)—イミディエイト・データ

CMP ♯<データ> TO <ea>

アセンブラ：CMPI ♯<データ>, <ea>

属性：サイズ=(バイト，ワード，ロングワード)

説明：イミディエイト・データをディスティネーション・オペランドから減算し，その結果に応じてコンディション・コードをセットします．ディスティネーションの位置の内容は変わりません．オペレーションのサイズはバイト，ワード，ロングワードに指定できます．イミディエイト・データのサイズはオペレーション・サイズに合致します．

コンディション・コード：

X	N	Z	V	C
—	*	*	*	*

N——結果が負であればセット，それ以外の時はクリア．

Z——結果が0であればセット，それ以外の時はクリア．

V——オーバーフローが発生すればセット，それ以外の時はクリア．

C——ボローが発生すればセット，それ以外の時はクリア．

X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	サイズ		実効アドレス					
ワード・データ(16ビット)								バイト・データ(8ビット)							
ロング・データ(32ビット、先行ワードを含む)															

命令フィールド：

サイズ・フィールド——オペレーション・サイズを指定．

00——バイト・オペレーション

01——ワード・オペレーション

10——ロングワード・オペレーション

実効アドレス・フィールド——ディスティネーションのオペランドを指定．データと可変のアドレッシング・モードのみ可．

イミディエイト・フィールド——（命令の直後のデータ）

サイズ=00，データはイミディエイト・ワードの下位バイト（8ビット）．

サイズ=01，データはイミディエイト・ワード（16ビット）．

サイズ=10，データはその次の2つのイディエイト・ワード（32ビット）．

CMPM (Compare Memory メモリ比較)

オペレーション：(ディスティネーション)—(ソース)

CMP Ay@+ TO A×@+

アセンブラ：CMPM (Ay)+, (Ax)+

属性：サイズ=(バイト, ワード, ロングワード)

説明：ソース・オペランドをディスティネーション・オペランドから減算し，その結果に応じてコンディション・コードをセットします．ディスティネーションの位置の内容は変化しません．オペランドは常に命令で指定されるアドレス・レジスタを用いた，ポストインクリメント・アドレッシング・モードでアドレスされます．オペレーションのサイズは，バイト，ワード，ローグワードに指定できます．

コンディション・コード：

X	N	Z	V	C
—	*	*	*	*

N——結果が負であればセット，それ以外の時はクリア．

Z——結果がゼロであればセット，それ以外の時はクリア．

V——オーバフローが発生すればセット，それ以外の時はクリア．

C——ボローが発生すればセット，それ以外の時はクリア．

X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	レジスタ Rx			1	サイズ		0	0	1	レジスタ Ry		

命令フィールド：

レジスタ Rx フィールド——ディスティネーションのアドレス・レジスタ（ポストインクリメント・アドレッシング・モードで使う）の番号を指定．

サイズ・フィールド——オペレーション・サイズを指定．

00——バイト・オペレーション

01——ワード・オペレーション

10——ロングワード・オペレーション

レジスタ Ry フィールド——ソースのアドレス・レジスタ（ポストインクリメント・アドレッシング・モードで使う）の番号を指定．

DBcc (Test Condition, Decrement and Branch 条件テスト，デクリメントとブランチ)

オペレーション：If～cc then Dn-1→Dn；if Dn ≒ 1 then PC+d→PC
CBcc Dn, <ラベル>

アセンブラ：DBcc Dn, <ラベル>

属性：サイズ＝(ワード)

説明：本命令は3つのパラメータ，すなわちコンディション，データ・レジスタおよびディスティネーションによるループ命令です．命令はまずコンディションをテストしてループ終了コンディション（termination condition）が満足されたかを判定し，もしそうであればブランチせず次の命令を実行します．もし終了コンディションが満足されない場合は，カウンタとして使用するデータ・レジスタの下位16ビットが1デクリメントされます．もし，その結果が，−1 なら，ブランチはせず次の命令を実行します．もし，その結果が −1 でなければ，実行は現在の PC 値に符号拡張された16ビット・ディスプレースメントを加えた位置から続行されます．PC 内の値はディスプレースメント・ワードのアドレスを示しています．“cc”としては次のコンディションを指定できます．

CC	0100—carry clear	LS	0011—low or same
CS	0101—carry set	LT	1101—less
EQ	0111—equal	MI	1011—minus
F	0001—never true	NE	0110—not equal
GE	1100—greater or equal	PL	1010—plus
GT	1110—greater	T	0000—always true
HI	0010—high	VC	1000—no overflow
LE	1111—less or equal	VS	1001—overflow

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化せず．
Z——変化せず．
V——変化せず．
C——変化せず．
X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	条件				1	1	0	0	1	レジスタ		
変位															

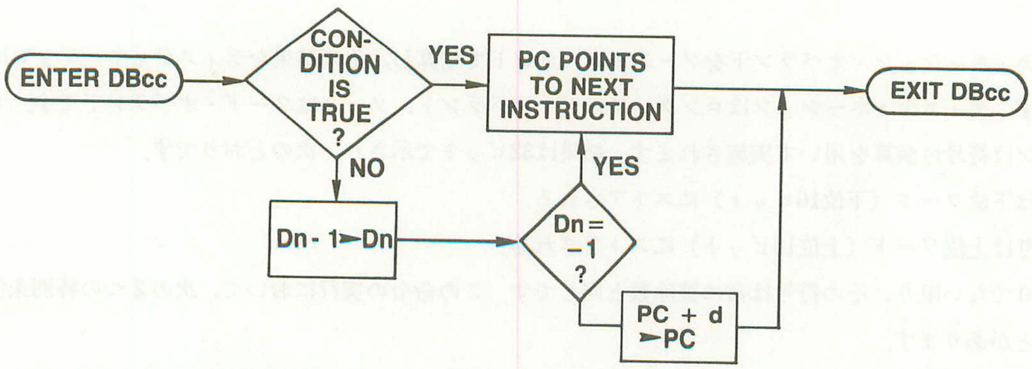
命令フィールド：

コンディション・フィールド——上記16コンディションの内の1つのコードを指定.

レジスタ・フールド——カウンタ・データ・レジスタの番号を指定.

ディスプレースメント——ブランチ先との相対距離をバイト単位で指定.

MC68,000 — DBcc INSTRUCTION



WHILE COUNT ≥ Ø AND TEST ≠ Ø DO
{ TEST }
COUNT: = COUNT −1
END OF WHILE LOOP

MC68,000 — DBcc EXAMPLES

AGAIN

•
•
•
•
•
•
•

		DØ.W	0001	DØ.W	0000	DØ.W	0000	REGISTER CONTENTS PRIOR TO EXECUTION
		SR	A600	SR	A604	SR	A600	
DBEQ	DØ, AGAIN	DØ.W: = 0; PC: = AGAIN		PC: = NEXT		DØ.W: = $FFFF; PC: = NEXT		RESULTS

NEXT

•
•
•

DIVS (Signed Divide　符号付き除算)

オペレーション：(ディスティネーション)／(ソース)→ディスティネーション
DIVS　Dn　BY <ea>

アセンブラ：DIVS <ea>, Dn

属性：サイズ＝(ワード)

説明：ディスティネーション・オペランドをソース・オペランドで除算し，その結果をディスティネーションにストアします．ディスティネーションはロング・ワード・オペランド，ソースはワード・オペランドです．オペレーションは符号付演算を用いて実施されます．結果は32ビットで示され，次のとおりです．

1．商は下位ワード（下位16ビット）にストアされる．
2．余りは上位ワード（上位16ビット）にストアされる．

余りが0でない限り，その符号は常に被除数と同じです．この命令の実行において，次の2つの特別条件が生じることがあります．

1．0による除算はトラップを発生し，エクセプション・プロセッシングを開始する．
2．命令完了前にもしオーバフローが検出されると，その条件はVに反映されるがオペランドは影響されない．

コンディション・コード：

X	N	Z	V	C
—	*	*	*	0

N――商が負であればセット，それ以外の時はクリア．ただし，オーバフローの場合は不定．
Z――商が0であればセット，それ以外の時はクリア．ただし，オーバフローの場合は不定．
V――除算のオーバフローが検出されるとセット，それ以外の時はクリア．
C――常にクリア．
X――変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	レジスタ			1	1	1	実効アドレス					

命令フィールド：

レジスタ・フィールド――データ・レジスタの番号を示し，常にディスティネーション・オペランド．
実効アドレス・フィールド――ソース・オペランドを指定．データ・アドレッシング・モードのみ可．

DIVU (Unsigned Divide 符号なし除算)

オペレーション：(ディスティネーション)/(ソース)→ディスティネーション
DIVU Dn By <ea>

アセンブラ：DIVU <ea>, Dn

属性：サイズ＝(ワード)

説明：ディスティネーション・オペランドをソース・オペランドで除算し，その結果をディスティネーションにストアします．ディスティネーション・オペランドはロングワード（32ビット），ソース・オペランドはワード（16ビット）です．オペレーションは符号なし演算を用いて行われます．結果は32ビットで示され，次のとおりです．

1．商は下位ワード（下位16ビット）にストアされる．
2．余りは上位ワード（上位16ビット）にストアされる．

この命令の実行において２つの特別条件が生じることがあります．

1．０による除算はトラップを発生し，エクセプション・プロセッシングを開始する．
2．命令完了前にもしオーバフローが検出されると，その条件はＶに反映されるが，オペランドは影響されない．

コンディション・コード：

X	N	Z	V	C
—	*	*	*	0

N――商の最上位ビットが１であればセット，それ以外の時はクリア．ただしオーバフローの場合は不定．
Z――商が０であればセット，それ以外の時はクリア．ただしオーバフローの場合は不定．
V――除算オーバフローが検出される場合セット，それ以外の時はクリア．
C――常にクリア．
X――変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	レジスタ			0	1	1	実効アドレス					

命令フィールド：

レジスタ・フィールド――データ・レジスタの番号を指定し，常にディスティネーション・オペランドです．
実効アドレス・フィールド――ソース・オペランドを指定．データ・アドレッシング・モードのみ可．

EOR (Exclusive OR Logical　論理 EOR)

オペレーション：(ディスティネーション)⊕(ソース)→ディスティネーション
EOR　Dn　TO <ea>

アセンブラ：EOR　Dn, <ea>

属性：サイズ＝(バイト，ワード，ロングワード)

説明：ソース・オペランドとディスティネーション・オペランドとの EOR をとり，結果をディスティネーションの位置に格納します．オペレーションのサイズはバイト，ワード，ロング・ワードに指定できます．このオペレーションのソース・オペランドは，データ・レジスタに限定されます．ソース・オペランドは実効アドレス・フィールドで指定されます．

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

N──結果の最上位ビットが1であればセット，それ以外の時はクリア．
Z──結果が0であればセット，それ以外の時はクリア．
V──常にクリア．
C──常にクリア．
X──変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	レジスタ			Op-モード			実効アドレス					

命令フィールド：

レジスタ・フィールド──データ・レジスタの番号を指定．

Op-モード・フィールド──

バイト	ワード	ロング・ワード	オペレーション
100	101	110	(<ea>)⊕(<Dx>)→<ea>

実効アドレス・フィールド──ディスティネーション・オペランドのアドレッシング・モードを指定します．データ可変のアドレッシング・モードのみ可．

EORI (Exclusive OR Immediate イミディエイト EOR)

オペレーション：（ディスティネーション）⊕ イミディエイト・データ→ディスティネーション

EORI ♯＜データ＞ TO ＜ea＞

アセンブラ：EORI ♯＜データ＞，＜ea＞

属性：サイズ＝(バイト，ワード，ロングワード)

説明：イミディエイト・データとディスティネーション・オペランドとの EOR をとり，結果をディスティネーションの位置に格納します．オペレーションのサイズはバイト，ワード，ロング・ワードに指定できます．イミディエイト・データはオペレーション・サイズに合致します．ステータス・レジスタへのアクセスは２つのモードで可能です．

1．もしオペレーション・サイズがバイトなら，このオペレーションはコンディション・コードを含むステータス・レジスタの下位バイトのみに対して実行される．

2．もしオペレーション・サイズがワードなら，オペレーションはステータス・レジスタ全体に対して実行される．これは特権操作．

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

N——結果の最上位ビットが１であればセット，それ以外の時はクリア．

Z——結果が０であればセット，それ以外の時はクリア．

V——常にクリア．

C——常にクリア．

X——変化せず．

ステータス・レジスタをアクセスする場合はそのオペレーションに従ってコンディション・コードが変化する．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	サイズ		実効アドレス					
ワード・データ（16ビット）								バイト・データ（8ビット）							
ロング・データ（32ビット、先行ワードを含む）															

命令フィールド：

サイズ・フィールド——オペレーション・サイズを指定．

00——バイト・オペレーション

01——ワード・オペレーション

10——ロングワード・オペレーション

実効アドレス・フィールド——ディスティネーションのアドレッシング・モードを指定する．データと可変のアドレッシング・モードおよびステータス・レジスタへのアドレッシング・モードのみ可．

イミディエイト・フィールド——（命令着直後のデータ）

サイズ＝00，データはイミディエイト・ワードの下位バイト（8ビット）．

サイズ＝01，データはイミディエイト・ワード（16ビット）．

サイズ＝10，データはその次の2つのイミディエイト・ワード（32ビット）．

EXG (Exchange Registers　レジスタ交換)

オペレーション：Rx⟷Ry

EXG　Rx　WITH　Ry

アセンブラ：EXG　Rx, Ry

属性：サイズ＝(ロングワード)

説明：2つのレジスタ内容を交換します．常にロングワード・オペレーションであり，次の3つのモードがあります．

1．データ・レジスタ間の交換
2．アドレス・レジスタ間の交換
3．データ・レジスタとアドレス・レジスタ間の交換

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化せず．
Z——変化せず．
V——変化せず．
C——変化せず．
X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	レジスタ Rx			1	Op-モード					レジスタ Ry		

命令フィールド：

レジスタRxフィールド——データ・レジスタまたはアドレス・レジスタの番号を指定．データ・レジスタとアドレス・レジスタ間の交換の場合は，このフィールドでは常にデータ・レジスタの番号を指定．

Op-モード・フィールド——交換するデータを指定．

01000——データ・レジスタ間．
01001——アドレス・レジスタ間．
10001——データ・レジスタとアドレス・レジスタ間．

レジスタRxフィールド——データ・レジスタまたはアドレス・レジスタの番号を指定．データ・レジスタとアドレス・レジスタ間の交換の場合は，このフィールドでは常にデータ・レジスタの番号を指定．

EXT (Sign Extend　符号拡張)

オペレーション：(ディスティネーション) Sign Extended→ディスティネーション
EXT　Dn

アセンブラ：EXT　Dn

属性：サイズ=(ワード，ロングワード)

説明：指定されたサイズに従い，データ・レジスタの符号ビットをバイトからワードまたはワードからロングワードに拡張します．もしオペレーション・サイズがワードなら，指定されたデータ・レジスタのビット〔7〕が，そのレジスタのビット〔15：8〕にコピーされます．もしオペレーション・サイズがロングワードなら，指定されたデータ・レジスタのビット〔15〕がそのデータ・レジスタのビット〔31：16〕にコピーされます．

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

N——結果が負であればセット，それ以外の時はクリア．
Z——結果が0であればセット，それ以外の時はクリア．
V——常にクリア．
C——常にクリア．
X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	Op-モード			0	0	0	レジスタ		

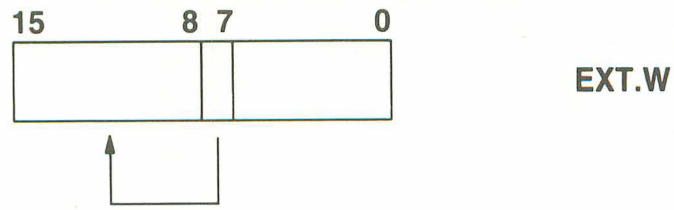
命令フィールド：

Op-モード・フィールド——符号拡張のオペレーション・サイズを指定．
010——バイトをワードへ拡張．
011——ワードをロング・ワードへ拡張．

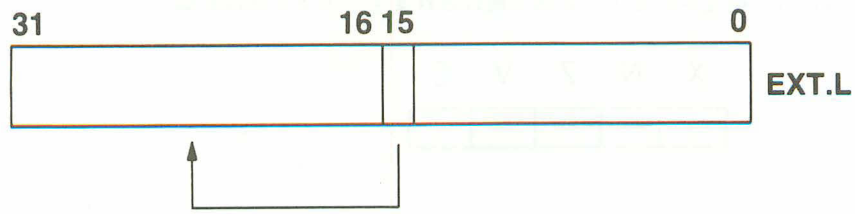
レジスタ・フィールド——符号拡張を行うデータ・レジスタを指定．

MC68,000 — EXT INSTRUCTION

BYTE TO WORD



WORD TO LONG WORD



JMP (Jump　ジャンプ)

オペレーション：ディスティネーション──→PC
JMP <ea>

アセンブラ：JMP <ea>

属性：サイズ不定

説明：プログラム実行は命令で指定されるアドレスで続行されます．アドレスは制御アドレッシング・モードで指定されます．

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N──変化せず．
Z──変化せず．
V──変化せず．
C──変化せず．
X──変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	1	実効アドレス					

命令フィールド：

実効アドレス・フィールド──ジャンプ先のアドレスを指定．制御アドレッシング・モードのみ可．

JSR (Jump to Subroutine サブルーチン・ジャンプ)

オペレーション： PC→SP @—；ディスティネーション→PC
JSR <ea>

アセンブラ： JSR <ea>

属性： サイズ不定

説明： JSR 命令の直後の命令のアドレスがシステム・スタックに格納されます．その後，プログラム実行は命令で指定されたアドレスで続行されます．

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化せず．
Z——変化せず．
V——変化せず．
C——変化せず．
X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	0	実効アドレス					

命令フィールド：

実効アドレス・フィールド——ジャンプ先のアドレスを指定．制御アドレッシング・モードのみ可．

LEA (Load Effective Address 実効アドレス・ロード)

オペレーション： ディスティネーション──→An

LEA <ea> INTO An

アセンブラ： LEA <ea>, An

属性： サイズ＝(ロングワード)

説明： ソース・オペランドの実行アドレスが指定されたアドレス・レジスタにロードされます．アドレス・レジスタの全32ビットがこの命令の影響を受けます．

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N──変化せず．
Z──変化せず．
V──変化せず．
C──変化せず．
X──変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	レジスタ			1	1	1	実効アドレス					

命令フィールド：

レジスタ・フィールド──実効アドレスをロードするアドレス・レジスタの番号を指定．

実効アドレス・フィールド──ソース・オペランドのアドレッシング・モードを指定．制御のアドレッシング・モードのみ可．

MC68,000 — LEA INSTRUCTION

REGISTER CONTENTS PRIOR TO EXECUTION	AØ 0 1 2 3 4 5 6 7 A1 X X X X X X X X	AØ 0 1 0 2 0 3 0 4 DØ 0 0 0 0 A 1 2 3 A1 X X X X X X X X	AØ 1 1 1 1 1 1 1 1 A1 2 2 2 2 2 2 2 2 A2 X X X X X X X X
INSTRUCTION	LEA (AØ), A1	LEA $10 (AØ,DØ), A1	LEA $91 (AØ,A1.L), A2
EXECUTION	0 1 2 3 4 5 6 7 → A 1	0 1 0 2 0 3 0 4 F F F F A 1 2 3 0 0 0 0 0 0 1 0 1 0 1 0 1 A 4 3 7 → A 1	1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 F F F F F F 9 1 1 3 3 3 3 3 2 C 4 → A 2
REGISTER CONTENTS AFTER EXECUTION	AØ 0 1 2 3 4 5 6 7 A1 0 1 2 3 4 5 6 7	AØ 0 1 0 2 0 3 0 4 DØ F F F F A 1 2 3 A1 0 1 0 1 A 4 3 7	AØ 1 1 1 1 1 1 1 1 A1 2 2 2 2 2 2 2 2 A2 3 3 3 3 3 2 C 4

CONTROL ADDRESSING MODES ARE ALLOWED

LINK (Link and Allocate リンクと割付け)

オペレーション： An→SP@－；SP→An；SP＋d→SP
LINK　An, ♯＜ディスプレースメント＞

アセンブラ： LINK　An, ♯＜ディスプレースメント＞

属性： サイズ不定

説明： 指定されたアドレス・レジスタの現在の内容がスタックに格納されます．その後アドレス・レジスタに更新された（プリデクリメント）スタック・ポインタ（Stack Pointer）の値がロードされます．最後に符号拡張されたディスプレースメントがスタック・ポインタに加えられます．アドレス・レジスタ内容はスタック内で2ワードを占めます．

（注） LINKとUNLKはネスティング（Nested）のサブルーチン・コールの時，スタック上のローカルのデータおよびパラメータ領域のリンク・リストを維持するのに使われます．

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N――変化せず．
Z――変化せず．
V――変化せず．
C――変化せず．
X――変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	0	レジスタ		
変位															

命令フィールド：

レジスタ・フィールド――リンクするためのアドレス・レジスタの番号を指定．
ディスプレースメント・フィールド――スタック・ポインタに加算する“2”の補数の整数．

MC68,000 — LINK AND UNLINK

HIGH LEVEL SOURCE CODE	COMPILER GENERATED CODE
A: PROCEDURE;	A:
•	•
•	•
•	•
CALL B (C);	PEA <EA> OF C
	JSR B
	LEA 4(SP), SP
•	•
•	•
•	•
END {A} ;	
B: PROCEDURE (X);	B: LINK FP, — PROCBALLOC
•	•
•	•
•	•
END {B} ;	UNLK FP
	RTS

MC68,000 — LINK, UNLINK EXAMPLE PROGRAM

```
LINKUN     MC68000 ASM REV= 1.0E- COPYRIGHT BY MOTOROLA 1978

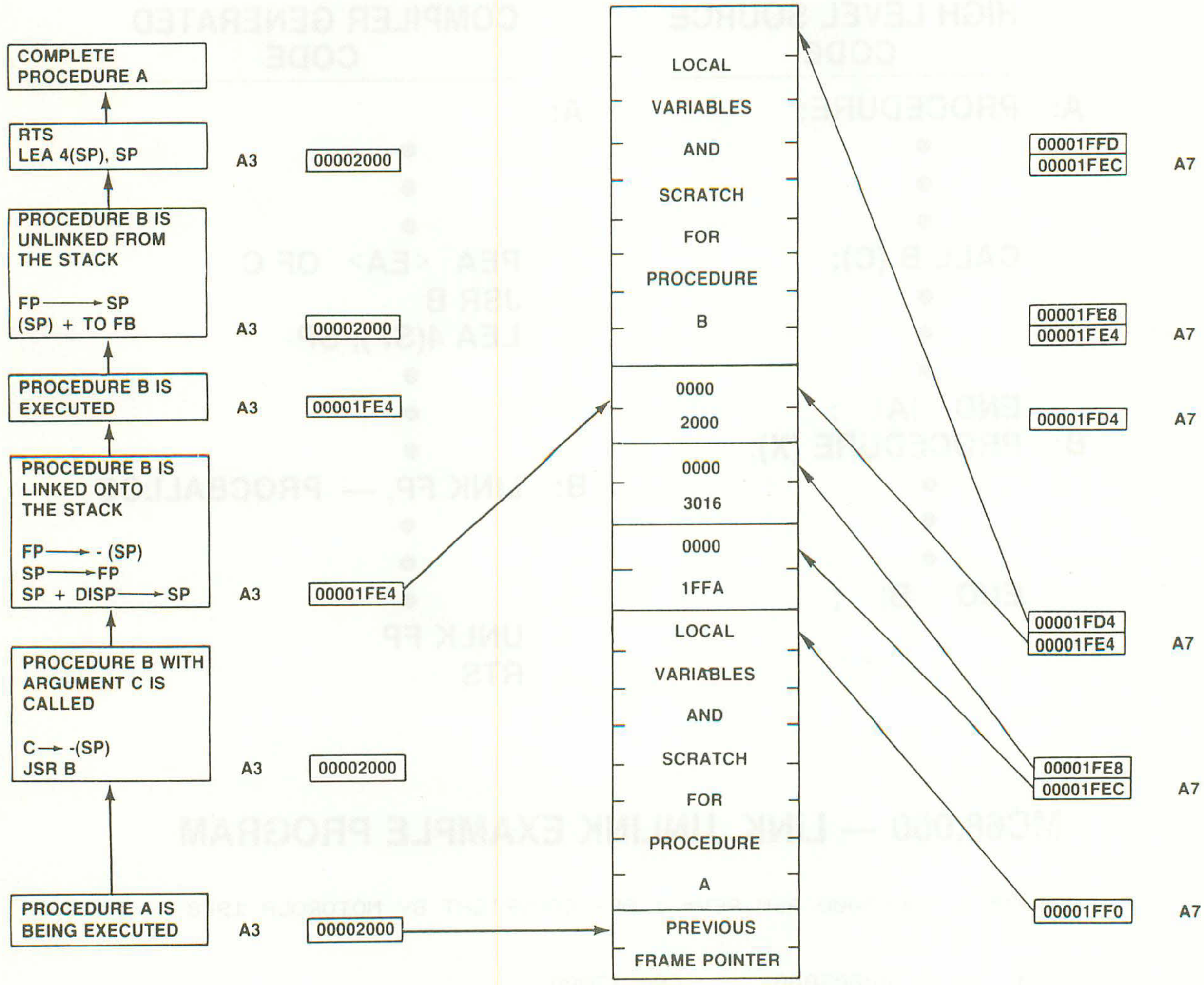
  1        00003000      ORG $3000
  2 003000 47F82000      LEA $2000,A3
  3 003004 4FF81FF0      LEA $1FF0,SP
  4 003008 4E71         PROCA NOP
  5 00300A 4E71          NOP
  6 00300C 4E71          NOP
  7 00300E 486BFFFA      PEA -6(A3)
  8 003012 4EB83022      JSR PROCB
  9 003016 4FEF0004      LEA 4(SP),SP
 10 00301A 4E71          NOP
 11 00301C 4E71          NOP
 12 00301E 4E71          NOP
 13 003020 60FE         ENDA BRA *
 14 003022 4E53FFF0     PROCB LINK A3,-$10
 15 003026 4E71          NOP
 16 003028 4E71          NOP
 17 00302A 4E71          NOP
 18 00302C 4E5B          UNLK A3
 19 00302E 4E75          RTS
 20                      END

****** TOTAL ERRORS   0--   0

SYMBOL TABLE

ENDA        003020 PROCA       003008 PROCB       003022
```

MC68,000 — LINK AND UNLINK



LSL, LSR (Logical Shift　論理シフト)

オペレーション：(ディスティネーション) Shifted dy ＜count＞→ディスティネーション

LSd　Dy　BY　Dx
LSd　Dy　BY　♯＜data＞
LSd　＜ea＞

アセンブラ：LSd　Dx, Dy
LSd ♯＜data＞, Dy
LSd ＜ea＞　(注) d は方向を示し，R または L である．

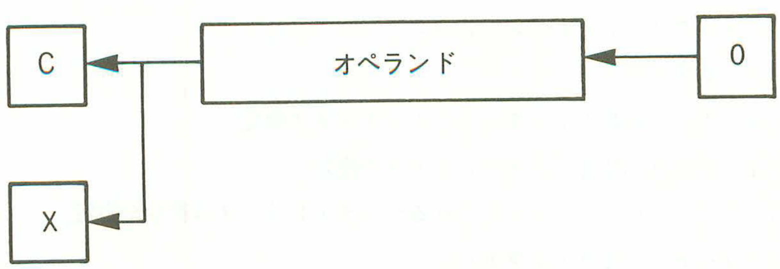
属性：サイズ＝(バイト，ワード，ロングワード)

説明：ディスティネーション・オペランドのビットを指定方向にシフトします．キャリおよび拡張ビットは，オペランドから最後に送り出されたビットを受け取ります．レジスタをシフトする場合のシフト回数（shift range）は2つの方法で指定できます．

1．イミディエイト：シフト回数は命令で指定できる．（1～8回）
2．レジスタ；シフト回数は命令で指定されるデータ・レジスタ内にある．

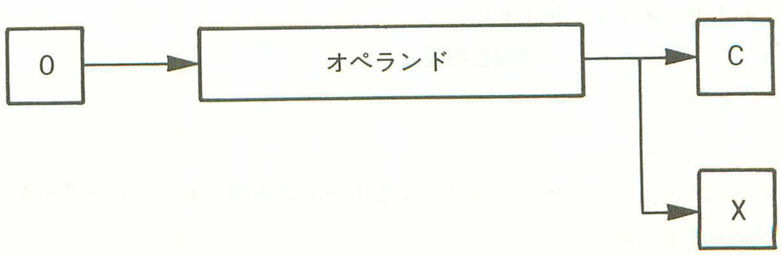
オペレーション・サイズはバイト，ワード，ロング・ワードに指定できます．ただし，メモリ内容の場合は1ビットしかシフトできず，オペレーション・サイズもワードに限定されます．
LSL では，オペランドは左シフトされ，シフトされた位置数がシフト回数です．最上位から送り出されたビットはキャリ，拡張の両ビットに入ります．最下位ビットは0が送り込まれます．

LSL:



LSR ではオペランドは右シフトされ，シフトされた位置数がシフト回数です．最下位から送り出されたビットはキャリと拡張の両ビットに入ります．最上位ビットには0が送り込まれます．

LSR:



コンディション・コード：

X	N	Z	V	C
*	*	*	0	*

N——結果が負であればセット，それ以外の時はクリア．

Z——結果が0であればセット，それ以外の時はクリア．

V——常にクリア．

C——オペランドから最後に送り出されたビットに従ってセット．シフト回数が0の時，クリア．

X——オペランドから最後に送り出されたビットに従ってセット，シフト回数が0の時は変化せず．

命令フォーマット：（レジスタ・シフト）：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	カウント/レジスタ			dr	サイズ		i/r	0	1	レジスタ		

命令フィールド（レジスタ・シフト）：

カウンタ／レジスタ・フィールド——

i/r=0，シフト回数を指定．値0，1～7がそれぞれ回数8，1～7をそれぞれ表す．

i/r=1，シフト回数はこのフィールドで指定されるデータ・レジスタ内にある．

dr　フィールド——シフト方向を指定．

0——右シフト．

1——左シフト．

サイズ・フィールド——オペレーションのサイズを指定．

00——バイト・オペレーション

01——ワード・オペレーション

10——ロングワード・オペレーション

i/r　フィールド——

i/r=0，シフト回数をイミディエイト・データで指定．

i/r=1，シフト回数をデータ・レジスタで指定．

レジスタ・フィールド——シフトされるデータ・レジスタの番号を指定．

命令フォーマット（メモリ・シフト）：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	1	dr	1	1	実効アドレス					

命令フィールド（メモリ・シフト）：

dr　フィールド——シフト方向を指定．

0——右シフト．

1——左シフト．

実効アドレス・フィールド——シフトするオペランドのアドレッシング・モードを指定．メモリと可変のアドレッシング・モードのみ可．

MOVE (Move Data from Source to Destination ソースから行き先へデータ移動)

オペレーション：(ソース)⟶ディスティネーション

MOVE <ea> TO <ea>

アセンブラ：MOVE <ea>, <ea>

属性：サイズ=(バイト，ワード，ロングワード)

説明：ソース・オペランドの内容をディスティネーションの位置に転送します．データは転送時に調べられ，コンディション・コードがセットされます．オペレーション・サイズはバイト，ワード，ロングワードに指定できます．

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

N——結果が負であればセット，それ以外の時はクリア．

Z——結果が0であればセット，それ以外の時はクリア．

V——常にクリア．

C——常にクリア．

X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				ディスティネーション						ソース					
0	0	サイズ		レジスタ			モード			モード			レジスタ		

命令フィールド：

サイズ・フィールド——転送するオペランドのサイズを指定．

01——バイト・オペレーション．

11——ワード・オペレーション．

10——ロングワード・オペレーション．

ディスティネーション実効アドレス・フィールド——ディスティネーションのアドレッシング・モードを指定．データ可変のアドレッシング・モードのみ可．

ソース実効アドレス・フィールド——ソース・オペランドのアドレッシング・モードを指定．すべてのアドレッシング・モードが可．ただし，オペレーション・サイズがバイトの場合は，直接アドレス・レジスタのアドレッシング・モードは不可．

MOVE to CCR (Move to Condition Codes CCR へ移動)

オペレーション：(ソース)→CCR
MOVE <ea> TO CCR

アセンブラ：MOVE <ea>, CCR

属性：サイズ=(ワード)

説明：ソース・オペランドの内容がコンディション・コードに移されます．ソース・オペランドはワードですが，下位バイトのみがコンディション・コードの更新に使われます．

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

N——ソース・オペランドに従ってセット．
Z——ソース・オペランドに従ってセット．
V——ソース・オペランドに従ってセット．
C——ソース・オペランドに従ってセット．
X——ソース・オペランドに従ってセット．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	1	1	実効アドレス					

命令フィールド：
実効アドレス・フィールド——ソース・オペランドのアドレッシング・モードを指定．データ・アドレッシング・モードのみ可．

MOVE to SR (Move to the Status Register SRへ移動)

オペレーション：(ソース)→SR
MOVE <ea> TO SR

アセンブラ：MOVE <ea>, SR

属性：サイズ＝(ワード)

説明：ソース・オペランドの内容がステータス・レジスタに移されます．ソース・オペランドはワードであり，SRの全ビットが影響をうけます．(注) これは特権命令です．

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

N——ソース・オペランドに従ってセット．
Z——ソース・オペランドに従ってセット．
V——ソース・オペランドに従ってセット．
C——ソース・オペランドに従ってセット．
X——ソース・オペランドに従ってセット．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	1	1	実効アドレス					

命令フィールド：

実効アドレス・フィールド——ソース・オペランドのアドレッシング・モードを指定．データ・アドレッシング・モードのみ可．

MOVE from SR (Move from the Status Register SRから移動)

オペレーション： SR→ディスティネーション
MOVE SR TO <ea>

アセンブラ： MOVE SR, <ea>

属性： サイズ＝(ワード)

説明： SR の内容がディスティネーションの位置に移されます．オペランド・サイズはワードです．

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N――変化せず．
Z――変化せず．
V――変化せず．
C――変化せず．
X――変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	1	実効アドレス					

命令フィールド：

実効アドレス・フィールド――ディスティネーション・オペランドのアドレッシング・モードを指定．データと可変のアドレッシング・モードのみ可．

MOVE USP (Move User Stack Pointer USP 移動)

オペレーション：USP→An；
An→USP
MOVE USP TO An
MOVE An TO USP

アセンブラ：MOVE USP, An
MOVE An, USP

属性：サイズ＝(ロングワード)

説明：ユーザ・スタック・ポインタの内容と指定されたアドレス・レジスタとを転送します．（注）これは特権命令です．

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化せず．
Z——変化せず．
V——変化せず．
C——変化せず．
X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	dr	レジスタ		

命令フィールド：

dr フィールド——転送の方向を指定．
0——アドレス・レジスタをユーザ・スタック・ポインタへ転送．
1——ユーザ・スタック・ポインタをアドレス・レジスタへ転送．

レジスタ・フィールド——アドレス・レジスタの番号を指定．

MOVEA (Move Address アドレス移動)

オペレーション：(ソース)→ディスティネーション

MOVE <ea> TO An

アセンブラ：MOVEA <ea>, An

属性：サイズ＝(ワード，ロングワード)

説明：ソース・オペランドの内容をディスティネーションのアドレス・レジスタに移します．オペレーション・サイズはワードかロングワードに指定できます．

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化せず．
Z——変化せず．
V——変化せず．
C——変化せず．
X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				変位						ソース					
0	0	サイズ		レジスタ			0	0	1	モード			レジスタ		

命令フィールド：

サイズ・フィールド——オペランドのサイズを指定．

11——ワード・オペレーション．ただし，ソース・オペランドはロングワードに符号拡張され，全32ビットがアドレス・レジスタにロードされる．

10——ロングワード・オペレーション

ディスティネーション・レジスタ・フィールド——ディスティネーションのアドレス・レジスタ番号を指定．

ソース実効アドレス・フィールド——ソース・オペランドのアドレッシング・モードを指定．すべてのアドレッシング・モードが可．

MOVEM (Move Multiple Registers　複数レジスタ移動)

オペレーション：レジスタ→ディスティネーション
(ソース)→レジスタ
MOVEM <レジスタ・リスト> TO <ea>
MOVEM <ea> TO <レジスタ・リスト>

アセンブラ：MOVEM <レジスタ・リスト>, <ea>
MOVEM <ea>, <レジスタ・リスト>

属性：サイズ=(ワード，ロングワード)

説明：レジスタ・リストで指定されたレジスタ群は，実効アドレスで指定した位置から始まる連続したメモリ位置へ，または，そのメモリ位置から転送されます．転送はマスク・フィールドにセットされているレジスタに関して行われます．オペレーションのサイズはロングワードまたはワードです．ワードをレジスタに転送する場合は，各ワードは32ビットに符号拡張され，ロングワードとしてレジスタにロードされます．

MOVEM は3種のアドレス・モード，すなわち制御のアドレッシング・モード，プリデクリメント・モードおよびポストインクリメント・モードのアドレッシングが可能です．実効アドレスが制御アドレッシング・モードであると，指定したメモリ位置から始まってアドレスの増加する方向に向かってレジスタ群との転送を行います．転送順序はデータ・レジスタ0からデータ・レジスタ7，次にアドレス・レジスタ0からアドレス・レジスタ7です．

もし実効アドレスがプリデクリメント・モードであれば，レジスタからメモリへの転送のみが可能です．レジスタは指定したメモリ位置－2（または4）に始まってアドレスの減少する方向へストアされます．ストア順序は上に述べた順序とは逆でアドレス・レジスタ7からアドレス・レジスタ0，次にデータ・レジスタ7からデータ・レジスタ0です．デクリメントされたアドレス・レジスタは最後にストアしたアドレスを示すように更新されます．

もし実効アドレスがポストインクリメントであれば，メモリからレジスタへの転送のみが許されます．レジスタは指定されたアドレスに始まり，アドレスの増加する方向のメモリからロードされます．ロードの順序は制御アドレッシング・モードの時と同じです．インクリメントされたアドレス・レジスタは，最後にロードしたアドレス＋2（または4）を示すように更新されます．

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化せず．
Z——変化せず．
V——変化せず．
C——変化せず．
X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	dr	0	0	1	Sz	実効アドレス					
レジスタ・リスト・マスク															

命令フィールド：

dr フィールド——転送の方向を指定.

0——レジスタからメモリ.

1——メモリからレジスタ.

Sz フィールド——転送のサイズを指定.

0——ワード・オペレーション

1——ロングワード・オペレーション

実効アドレス・フィールド——メモリ・アドレッシング・モードを指定.

レジスタからメモリへの転送の場合は，制御と可変のアドレッシング・モードまたはプリデクリメント・アドレッシング・モードのみ可.

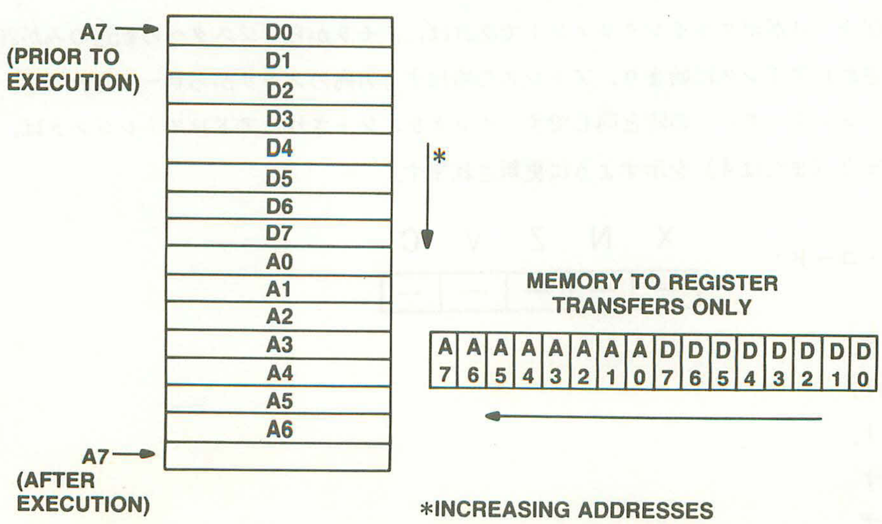
メモリからレジスタへの転送の場合は，制御アドレッシング・モードまたはポストインクリメント・アドレッシング・モードのみ可.

レジスタ・リスト・マスク・フィールド——転送に使用するレジスタを指定．下位ビットは最初に転送するレジスタに対応し，上位ビットは最後に転送するレジスタに対応．コントロールおよびポストインクリメント・アドレッシング・モードの場合，レジスタは下記のビットに対応.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0

MC68,000 — MOVEM (POST INCREMENT)

4C9F7FFF MOVEM (A7)+,A0 – A6/D0-D7

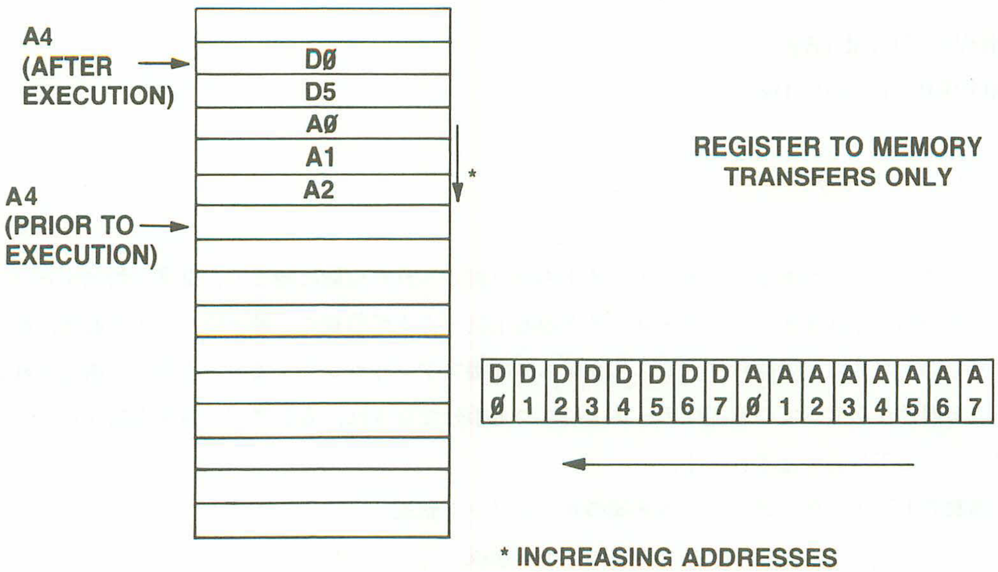


プリデクリメント・アドレッシング・モードの場合，レジスタは下記のビットに対応.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D0	D1	D2	D3	D4	D5	D6	D7	A0	A1	A2	A3	A4	A5	A6	A7

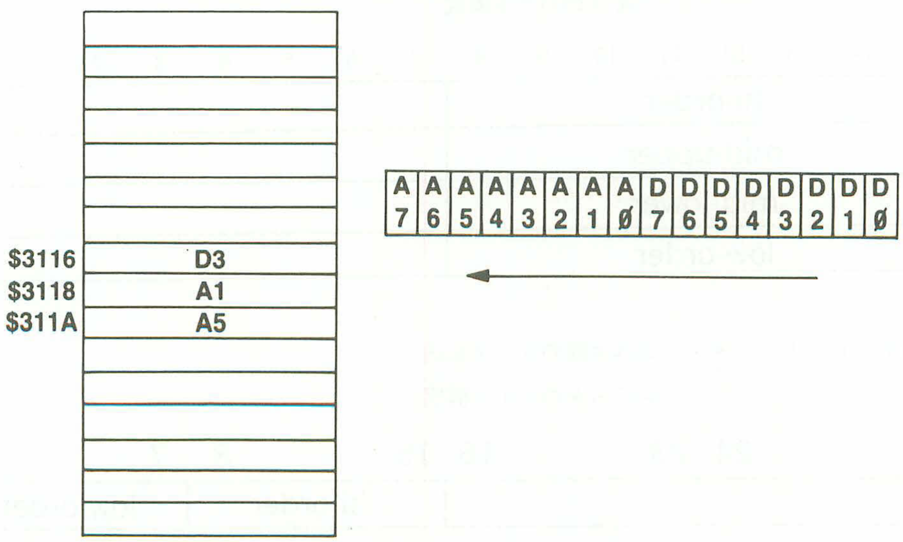
MC68,000 — MOVEM (PREDECREMENT)

48A184EØ MOVEM DØ/D5/AØ – A2, –(A4)



MC68,000 — MOVEM (REGISTERS TO MEMORY)

48B822Ø83116 MOVEM A1/A5/D3, $3116



MOVEP (Move Peripheral Data　周辺データ移動)

オペレーション：(ソース)→ディスティネーション

MOVEP　Dx　TO　Ay@（d）

MOVEP　Ay@（d）　TO　Dx

アセンブラ：MOVE　Dx, d（Ay）

MOVEP　d（Ay）, Dx

属性：サイズ＝(ワード，ロングワード)

説明：データ・レジスタと，1つおきのメモリ・バイトの間（指定された位置に始まり，2ずつ増加するメモリ位置）でデータが転送されます．データ・レジスタの最上位バイトが最初に，最下位バイトが最後に転送されます．メモリのアドレスは，変位付アドレス・レジスタ間接アドレッシング・モードを用いて指定されます．もしアドレスが偶数ならすべての転送はデータ・バスの上半分でなされ，もしアドレスが奇数ならすべての転送はデータ・バスの下半分でなされます．

<例>偶数アドレスとデータ・レジスタ間のロングワード転送．

レジスタのバイト構成

31　　24	23　　16	15　　8	7　　0
hi-order	mid-upper	mid-lower	low-order

メモリのバイト構成

15　14　13　12　11　10　9　8	7　6　5　4　3　2　1　0
hi-order	
mid-upper	
mid-lower	
low-order	

<例>奇数アドレスとデータ・レジスタ間のワード転送

レジスタのバイト構成

31　　24	23　　16	15　　8	7　　0
		hi-order	low-order

メモリのバイト構成

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								hi-order							
								low-order							

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化せず.

Z——変化せず.

V——変化せず.

C——変化せず.

X——変化せず.

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	データレジスタ			Op-モード			0	0	1	アドレスレジスタ		
変位															

命令フィールド：

データ・レジスタ・フィールド——データ・レジスタの番号を指定.

Op-モード・フィールド——オペレーションのサイズと方向を指定.

100——メモリからレジスタへワードを転送.

101——メモリからレジスタへロングワードを転送.

110——レジスタからメモリへワードを転送.

111——レジスタからメモリへワードを転送.

アドレス・レジスタ・フィールド——アドレス・レジスタの番号を指定. このアドレス・レジスタは変位付アドレス・レジスタ間接アドレッシング・モードに使用.

ディスプレースメント・フィールド——オペランド・アドレスを計算するのに使用するディスプレースメントを指定.

MOVEQ (Move Quick クイック移動)

オペレーション：イミディエイト・データ→ディスティネーション

MOVEQ ♯<data>TO Dn

アセンブラ：MOVEQ ♯<data>, Dn

属性：サイズ＝(ロングワード)

説明：イミディエイト・データをディスティネーションのデータ・レジスタに移します．データはオペレーション・ワード内の下位8ビット・フィールドにあります．データはロングワードに符号拡張され，全32ビットがデータ・レジスタに転送されます．

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

N——結果が負であればセット，それ以外の時はクリア．

Z——結果が0であればセット，それ以外の時はクリア．

V——常にクリア．

C——常にクリア．

X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	レジスタ			0	データ							

命令フィールド：

レジスタ・フィールド——データ・レジスタの番号を指定．

データ・フィールド——ロングワードに符号拡張した8ビットのイミディエイト・データ．

MULS (Signed Multipy 符号付き乗算)

オペレーション：（ディスティネーション）*（ソース）→ディスティネーション
MULS Dn By <ea>

アセンブラ：MULS <ea>, Dn

属性：サイズ（ワード）

説明：2個の符号付き16ビット整数のオペランドを乗算して32ビットの符号付きの結果を得ます．オペレーションは符号付き演算を用いて行われます．レジスタのオペランドは下位ワードを使用し，上位ワードは使われません．積は32ビットでディスティネーションのデータ・レジスタに入ります．

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

N——結果が負であればセット，それ以外のときクリア．
Z——結果が0であればセット，それ以外のときクリア．
V——常にクリア
C——常にクリア
X——変化せず

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	レジスタ			1	1	1	実効アドレス					

命令フィールド：

レジスタ・フィールド・ディスティネーションのデータ・レジスタ番号を指定．

実効アドレス・フィールド——ソース・オペランド・アドレッシング・モードを指定．データ・アドレッシング・モードのみ可．

MULU (Unsigned Multiply 符号なし乗算)

オペレーション：(ディスティネーション)*(ソース)→ディスティネーション
MULU Dn By <ea>

アセンブラ：MULU <ea>, Dn

属性：サイズ＝(ワード)

説明： 2個の符号なし16ビット整数のオペランドを乗算し32ビットの符号なし結果を得ます．オペレーションは符号なし演算を用いて行われます．レジスタのオペランドは下位ワードを使用し上位ワードは使われません．積は32ビットでディスティネーション・データ・レジスタに入ります．

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

N——結果の最上位ビットが1であればセット，それ以外の時クリア．
Z——結果が0であればセット，それ以外の時クリア．
V——常にクリア．
C——常にクリア．
X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	レジスタ			0	1	1	実効アドレス					

命令フィールド：

レジスタ・フィールド——ディスティネーションのデータ・レジスタ番号を指定．
実効アドレス・フィールド——ソース・オペランドのアドレッシング・モードを指定します．アドレッシング・モードのみ可

NBCD (Negate Decimal with Extend 拡張付き10進補数)

オペレーション： 0 — (ディスティネーション)$_{10}$ — X → ディスティネーション
NBCD <ea>

アセンブラ： NBCD <ea>

属性： サイズ＝(バイト)

説明： ディスティネーション・オペランドと拡張ビットを0から減算します．オペレーションは10進演算を用いて行います．結果はディスティネーションの位置に入れられます．このオペレーションはもし拡張ビットがなければディスティネーションの“10”の補数，もし拡張ビットがセットされていれば“9”の補数を作ります．これはバイト・オペレーションのみです．

コンディション・コード：

X	N	Z	V	C
*	U	*	U	*

N——不定
Z——結果が0でなければクリア，それ以外の時は変化せず．
V——不定
C——キャリ（10進）が発生したらセット，それ以外の時はクリア．
X——キャリ・ビットと同じ．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	0	実効アドレス					

命令フィールド：

実効アドレス・フィールド——ディスティネーション・オペランドのアドレッシング・モードを指定．データと可変のアドレッシング・モードのみ可．

NEG (Negate 補数)

オペレーション： 0－(ディスティネーション)→ディスティネーション
NEG <ea>

アセンブラ： NEG <ea>

属性： サイズ（バイト，ワード，ロングワード）

説明： ディスティネーションとしてアドレスされたオペランドが0から減算されます．結果はディスティネーションの位置にストアされます．オペレーションのサイズは，バイト，ワード，ロングワードに指定できます．

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

N——結果が負であればセット，それ以外のときクリア．
Z——結果が0であればセット，それ以外のときクリア．
V——オーバフローが発生したときセット，それ以外のときクリア．
C——ボローが発生したときセット．それ以外のときクリア．
X——キャリ・ビットと同様．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	サイズ		実効アドレス					

命令フィールド：

サイズ・フィールド——オペレーション・サイズを指定します．
00——バイト・オペレーション
01——ワード・オペレーション
10——ロングワード・オペレーション
実効アドレス・フィールド——ディスティネーション・オペランドのアドレッシング・モードを指定．データと可変のアドレッシング・モードのみ可．

NEGX (Negate with Extend 拡張付き補数)

オペレーション： 0—(ディスティネーション)—X→ディスティネーション
NEGX <ea>

アセンブラ： NEGX <ea>

属性： サイズ＝(バイト，ワード，ロングワード)

説明： ディスティネーションとしてアドレスされたオペランドと拡張ビットを0から減算します．結果はディスティネーションの位置にストアされます．オペレーションのサイズはバイト，ワード，ロングワードに指定できます．

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

N——結果が負であればセット，それ以外のときクリア．
Z——結果が0であればセット，それ以外のときクリア．
V——オーバフローが発生したときセット，それ以外のときクリア．
C——ボローが発生したときセット，それ以外のときクリア．
X——キャリ・ビットと同じ．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	サイズ		実効アドレス					

命令フィールド：

サイズ・フィールド——オペレーション・サイズを指定します．
- 00——バイト・オペレーション
- 01——ワード・オペレーション
- 10——ロングワード・オペレーション

実効アドレス・フィールド——ディスティネーション・オペランドのアドレッシング・モードを指定．データと可変のアドレッシング・モードのみ可．

NOP (No Operation 無操作)

オペレーション：

アセンブラ：NOP

属性：サイズなし

説明：何もしません．プロセッサ状態はPCを除き変化しません．実行はNOP命令の次の命令から続行されます．

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化なし
Z——変化なし
V——変化なし
C——変化なし
X——変化なし

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1

NOT (Logical Complement 論理否定)

オペレーション：～(ディスティネーション)→ディスティネーション

NOT <ea>

アセンブラ：NOT <ea>

属性：サイズ＝(バイト，ワード，ロングワード)

説明：ディスティネーション・オペランドの“1”の補数をとり結果をディスティネーションの位置にストアします．オペレーションのサイズは，バイト，ワード，ロングワードに指定できます．

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

N——結果が負であればセット，それ以外のときクリア．

Z——結果が0であればセット，それ以外のときクリア．

V——常にクリア．

C——常にクリア．

X——変化なし．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	サイズ		実効アドレス					

命令フィールド：

サイズ・フィールド——オペレーション・サイズを指定する．

00——バイト・オペレーション

01——ワード・オペレーション

10——ロング・ワード・オペレーション

実効アドレス・フィールド——ディスティネーション・オペランドのアドレッシング・モードを指定．データと可変のアドレッシング・モードのみ可．

OR (Inclusive OR Logical; 論理インクルーシブ OR)

オペレーション：(ディスティネーション)∨(ソース)→ディスティネーション

OR <ea> TO Dn

OR Dn TO <ea>

アセンブラ：OR <ea>, Dn

OR Dn, <ea>

属性：サイズ＝(バイト，ワード，ロングワード)

説明：ソース・オペランドとディスティネーション・オペランドとの論理 OR をとり，結果をディスティネーションの位置にストアします．オペレーションのサイズは，バイト，ワード，ロングワードに指定できます．アドレス・レジスタの内容はオペランドとして使用できません．

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

N——結果の最上位ビットが1であればセット，それ以外の時，クリア．

Z——結果が0であればセット，それ以外の時，クリア．

V——常にクリア．

C——常にクリア．

X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	レジスタ			Op-モード			実効アドレス					

命令フィールド：

レジスタ・フィールド——8データ・レジスタの番号を指定．

Op-モード・フィールド

バイト	ワード	ロングワード	オペレーション
000	001	010	(<Dn>)∨(<ea>)→<Dn>
100	101	110	(<ea>)∨(Dn)→<ea>

実効アドレス・フィールド——アドレッシング・モードを指定．

ソース・オペランドが指定された場合は，データ・アドレッシング・モードのみ可．

ディスティネーション・オペランドが指定された場合は，可変とメモリのアドレッシング・モードのみ可．

ORI (Inclusive OR Immediate イミディエイト・インクルーシブ OR)

オペレーション：(ディスティネーション)∨イミディエイト・データ→ディスティネーション

OR ♯<データ> TO <ea>

アセンブラ：ORI ♯<データ>, <ea>

属性：サイズ=(バイト，ワード，ロング・ワード)

説明 イミディエイト・データとディスティネーション・オペランドとの論理 OR をとり，結果をディスティネーションの位置にストアします．オペレーション・サイズはバイト，ワード，ロングワードに指定できます．イミディエイト・データのサイズはオペレーション・サイズに合致します．SR へのアクセスは次の 2 つのモードで可能です．

1 オペレーション・サイズがバイトなら，オペレーションはコンディション・コードを含む SR の下位バイトのみに対して実行される．

2 オペレーション・サイズがワードなら，オペレーションは SR 全体に対して実行される．これは特権操作である．

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

N——結果の最上位ビットが 1 であればセット，それ以外の時，クリア

Z——結果が 0 であればセット，それ以外の時，クリア

V——常にクリア．

C——常にクリア．

X——変化せず．

ステータス・レジスタをアクセスする場合は，そのオペレーションに従ってコンディション・コードが変化する．

命令フォーマット：

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="2">サイズ</td><td colspan="6">実効アドレス</td></tr>
<tr><td colspan="8">ワード・データ(16ビット)</td><td colspan="8">バイト・データ(8ビット)</td></tr>
<tr><td colspan="16">ロング・データ(32ビット、先行ワードを含む)</td></tr>
</table>

命令フィールド：

サイズ・フィールド——オペレーション・サイズを指定．

00——バイト・オペレーション

01——ワード・オペレーション

10——ロングワード・オペレーション

実効アドレス・フィールド——ディスティネーションのアドレッシング・モードを指定．データと可変のアドレッシング・モードおよびステータス・レジスタへのアドレッシング・モードのみ可．

イミディエイト・フィールド——命令の直後のデータ

サイズ＝00なら，データはイミディエイト・ワードの下位バイト（8ビット）

サイズ＝01なら，データはイミディエイト・ワード（16ビット）

サイズ＝10なら，データは次の2つのイミディエイト・ワード（32ビット）

PEA (Push Effective Address 実行アドレス格納)

オペレーション：ディスティネーション→SP@—
PEA <ea>

アセンブラ：PEA <ea>

属性：サイズ＝(ロングワード)

説明：実効アドレスが計算されロングワードでスタックに格納されます.

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化せず
Z——変化せず
V——変化せず
C——変化せず
X——変化せず

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	実効アドレス					

命令フィールド：

実効アドレス・フィールド——スタックに格納されるアドレスのアドレッシング・モードを指定. 制御アドレッシング・モードのみ可.

RESET (Reset External Devices 外部デバイス・リセット)

オペレーション：

アセンブラ： RESET

属性： サイズなし

説明： リセット出力をアサートし，すべての外部デバイスをリセットします．PC 以外のプロセッサ状態は変化せず，実行は次の命令から続行されます．（注）これは特権命令です．

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化せず
Z——変化せず
V——変化せず
C——変化せず
X——変化せず

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	0

ROL ROR (Rotate without Extend 拡張なしローテイト)

オペレーション：(ディスティネーション) Rotated by <カウント>→ディスティネーション

ROd Dy By Dx

ROd Dy By ♯ <データ>

ROd <ea>　　　(注) d は方向を表し L または R です)

アセンブラ：ROd Dx, Dy

ROd ♯ <データ>, Dy

ROd <ea>

属性：サイズ=(バイト，ワード，ロングワード)

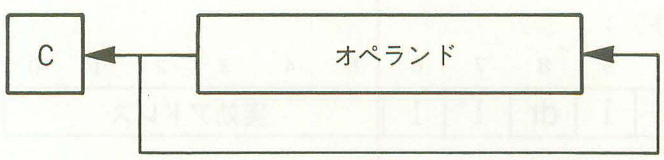
説明：ディスティネーション・オペランドのビットを指定方向にローテイト（回転）させます．拡張ビットは回転に含まれません．レジスタの・ローテイト回数は2つの方法で指定できます．

1　イミディエイト―ローテイト回数は指定される．（1～8回）

2　レジスター―ローテイト回数は命令で指定されるデータ・レジスタ内にある．オペレーションのサイズは，バイト，ワード，ロングワードに指定できる．ただし，メモリ内容の場合は1ビットしかローテイトできず，オペランド・サイズもワードに限定される．

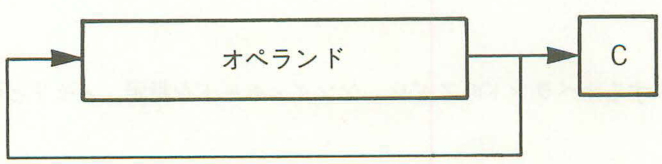
ROL では，オペランドは左ローテイト（回転）されます．シフトされた位置数がローテイト回数になります．最上位から送出されたビットはキャリ・ビットと最下位ビットの両方に入ります．拡張ビットは変化しません．

ROL



ROR では，オペランドは右ローテイト（回転）されます．シフトされた位置数がローテイト回数になります．最下位ビットから送出されたビットは，キャリ・ビットと最上位ビットの両方に入ります．拡張ビットは変化しません．

ROR



コンディション・コード：

X	N	Z	V	C
—	*	*	0	*

N――結果の最上位ビットが1であればセット，それ以外の時はクリア．

Z――結果が0であればセット，それ以外の時はクリア．

V――常にクリア．

C――オペランドから最後に送り出されたビットに従ってセット．シフト回数が“0”の時にクリア．

X――変化せず．

命令フォーマット（レジスタ・ローテイト）：

15	14	13	12	11 10 9	8	7 6	5	4	3	2 1 0
1	1	1	0	カウント/レジスタ	dr	サイズ	i/r	1	1	レジスタ

命令フィールド（レジスタ・ローテイト）：

カウンタ／レジスタ・フィールド――

i/r=0，ローテイト回数を指定．値0，1－7がそれぞれローテイト回数8，1～7を表す．

i/r=1，ローテイト回数は，このフィールドで指定されるデータ・レジスタ内にある．

dr フィールド――ローテイトの方向を指定．

0――右ローテイト

1――左ローテイト

サイズ・フィールド――オペレーション・サイズを指定する．

00――バイト・オペレーション

01――ワード・オペレーション

10――ロングワード・オペレーション

i/r フィールド――

i/r=0，ローテイト回数をイミディエイト・データで指示．

i/r=1，ローテイト回数をデータ・レジスタで指定．

レジスタ・フィールド――ローテイトされるデータ・レジスタの番号を指定．

命令フォーマット（メモリ・ローテイト）：

15	14	13	12	11	10	9	8	7	6	5 4 3 2 1 0
1	1	1	0	0	1	1	dr	1	1	実効アドレス

命令フィールド（メモリ・ローテイト）：

dr フィールド――ローテイト方向を指定．

0――右ローテイト

1――左ローテイト

実効アドレス・フィールド――ローテイトするオペランドのアドレッシング・モードを指定．メモリと可変のアドレッシング・モードのみ可．

ROXL, ROXR (Rotate with Extend 拡張付ローテイト)

オペレーション：(ディスティネーション) Rotated by <カウント>→ディスティネーション

ROXd Dy BY Dx

ROXd Dy By ♯<データ>

ROXd <ea>

アセンブラ：ROXd Dx, Dy

ROXd Dx, Dy

ROXd ♯<データ>, Dy

ROXd <ea>

属性：サイズ＝(バイト，ワード，ロングワード)

説明：ディスティネーション・オペランドのビットを指定方向にローテイト回転させます．拡張ビットがローテイトに含まれます．レジスタのローテイトの回数は2つの方法で指定できます．

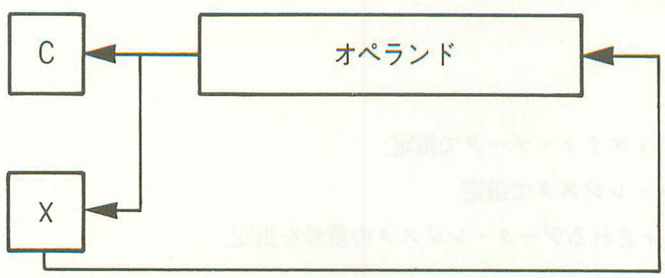
1　イミディエイト：ローテイト回数は，命令で指定される．（1～8回）

2　レジスタ：ローテイト回数は命令で指定されるデータ・レジスタ内にある．

オペレーションのサイズは，バイト，ワード，ロングワードに指定できます．ただし，メモリ内容の場合は1ビットしか回転できず，オペランド・サイズもワードに限定されます．

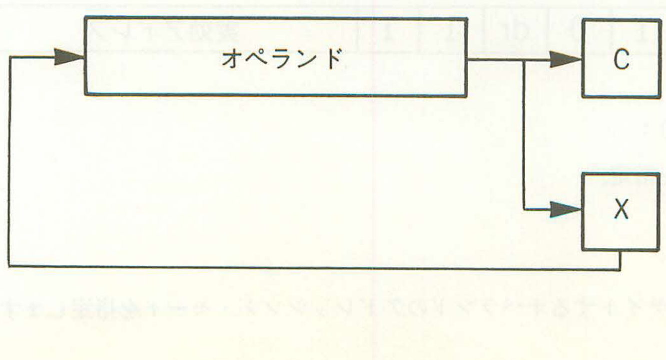
ROXL では，オペランドは左ローテイトされ，シフトされた位置数がローテイト回数です．最上位ビットから送出されたビットはキャリと拡張の両ビットに入り，拡張ビットの前の値が最下位ビットに送り込まれます．

ROXL



ROXR では，オペランドは右ローテイトされ，シフトされた位置数がローテイト回数です．最下位ビットから送出されたビットはキャリと拡張の両ビットに入り，拡張ビットの前の値が最上位ビットに送り込まれます．

ROXR



コンディション・コード：

X	N	Z	V	C
*	*	*	0	*

N——結果の最上位ビットが1であればセット，それ以外の時はクリア．

Z——結果が0であればセット，それ以外の時はクリア．

V——常にクリア．

C——オペランドから最後に送り出されたビットに従ってセット．ローテイト回数が0のときはXビットと同様にセット．

X——オペランドから最後に送り出されたビットに従ってセット．シフト回数が0のときは変化せず．

命令フォーマット（レジスタ・ローテイト）：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	カウント/レジスタ			dr	サイズ		i/r	1	0	レジスタ		

命令フィールド（レジスタ・ローテイト）：

カウンタ／レジスタ・フィールド：

i/r＝0，ローテイト回数を指定．値0，1—7がそれぞれローテイト回数8，1～7を表す．

i/r＝1，ローテイト回数はこのフィールドで指定されるデータ・レジスタ内にある．

dr フィールド——ローテイトの方向を指定．

0——右ローテイト

1——左ローテイト

サイズ・フィールド——オペレーション・サイズを指定します．

00——バイト・オペレーション

01——ワード・オペレーション

10——ロングワード・オペレーション

i/r フィールド——

i/r＝0，ローテイト回数をイミディエイト・データで指定．

i/r＝1，ローテイト回数をデータ・レジスタで指定．

レジスタ・フィールド——ローテイトされるデータ・レジスタの番号を指定．

命令フォーマット（メモリ・ローテイト）：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	1	0	dr	1	1	実効アドレス					

命令フィールド（メモリ・ローテイト）：

dr フィールド——ローテイト方向を指定．

0——右ローテイト

1——左ローテイト

実効アドレス・フィールド——ローテイトするオペランドのアドレッシング・モードを指定します．メモリと可変

のアドレッシング・モードのみ可.

RTE (Return from Exception　エクセプションからのリターン)

オペレーション：SP@+→SR；SP@+→PC
RTE

アセンブラ：RTE

属性：サイズなし

説明：SR と PC をスタックからとり出します．命令実行前の SR と PC は失われます．SR と全ビットが影響されます．とり出したプログラム・カウンタの示すアドレスから処理を続行します．

（注）これは特権命令であり，エクセプション・プロセッシングを開始時にスタックした内容をとり出しエクセプション・プロセッシングを開始する前の状態に戻ります．

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

N――スタックからのワード内容に従ってセットされます．
Z――スタックからのワード内容に従ってセットされます．
V――スタックからのワード内容に従ってセットされます．
C――スタックからのワード内容に従ってセットされます．
X――スタックからのワード内容に従ってセットされます．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

RTR (Return and Restore Condition Codes リターンと条件コード復元)

オペレーション：SP@+→CC; SP@+→PC
RTR

アセンブラ：RTR

属性：サイズなし

説明：コンディション・コードと PC をスタックからとり出します．命令実行前のコンディション・コードと PC は失われます．SR のスーパバイザ部（上位 8 ビット）は変化しません．取り出したプログラム・カウンタの示すアドレスから処理を続行します．

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

N——スタックからのワード内容に従ってセット．
Z——スタックからのワード内容に従ってセット．
V——スタックからのワード内容に従ってセット．
C——スタックからのワード内容に従ってセット．
X——スタックからのワード内容に従ってセット．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1

RTS (Return from Subroutine サブルーチンからリターン)

オペレーション：SP@+→PC
RTS

アセンブラ：RTS

属性：サイズなし

説明　PC をスタックからとり出します．命令実行前の PC は失われます．とり出したプログラム・カウンタの示すアドレスから処理を続行します．

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化なし
Z——変化なし
V——変化なし
C——変化なし
X——変化なし

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

SBCD (Subtract Decimal with Extend 拡張10進減算)

オペレーション：(ディスティネーション)$_{10}$—(ソース)$_{10}$—X→ディスティネーション
SBCD Dy FROM Dx
SBCD Ay@—FROM Ax@—

アセンブラ：SBCD Dy, Dx
SBCD —(Ay),—(Ax)

属性：サイズ=(バイト)

説明：ソース・オペランドと拡張ビットをディスティネーション・オペランドから減算し，結果をディスティネーションの位置にストアします．減算は，BCD 演算を用いて行われます．オペランドは，2つの方法でアドレスできます．

1 データ・レジスタからデータ・レジスタ： オペランドは命令で指定されるデータ・レジスタ内にある．
2 メモリからメモリ： オペランドは，命令で指定されるアドレス・レジスタを用いたプリデクリメント・アドレッシング・モードでアドレスされます．

コンディション・コード：

X	N	Z	V	C
*	U	*	U	*

N——不定.
Z——結果が0でなければクリア，それ以外のとき変化せず.
V——不定.
C——キャリ（10進）が発生したらセット，それ以外のときクリア.
X——キャリ・ビットと同じ.

命令フォーマット：

15	14	13	12	11 10 9	8	7	6	5	4	3	2 1 0
1	0	0	0	レジスタ Rx	1	0	0	0	0	R/M	レジスタ Ry

命令フィールド：

レジスタ Rx フィールド——ディスティネーションのレジスタ番号
R/M=0，データ・レジスタの番号
R/M=1，アドレス・レジスタ（プリデクリメント・アドレス・モードで使用）の番号
R/M フィールド——オペランドのアドレッシング・モードを指定
0——データ・レジスタからデータ・レジスタ

1——メモリからメモリ

レジスタ Ry フィールド——ソース・レジスタ番号を指定.

R/M=0, データ・レジスタの番号

R/M=1, アドレス・レジスタ（プリデクリメント・アドレス・モードで使用）の番号

Scc (Set According to Condition コンディション・セット)

オペレーション：If cc then 1's →ディスティネーション else 0's →ディスティネーション
Scc <ea>

アセンブラ：Scc <ea>

属性：サイズ＝(バイト)

説明：指定したコンディション・コードがテストされ，コンディションがもし真なら実効アドレスで指定されるバイトが真（すべて1）にセットされ，そうでなければ偽（すべて0）にセットされます．“cc”として，以下のコンディションを指定できます．

CC — carry clear	LS — low or same
CS — carry set	LT — less
EQ — equal	MI — minus
F — never true	NE — not equal
GE — greater or equal	PL — plus
GT — greater	T — always true
HI — high	VC — no overflow
LE — less or equal	VS — overflow

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化せず．
Z——変化せず．
V——変化せず．
C——変化せず．
X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	条件				1	1	実効アドレス					

命令フィールド：

コンディション・フィールド——次の16コンディションのいずれかを指定します．

CC — 0100	LS — 0011
CS — 0101	LT — 1101
EQ — 0111	MI — 1011

F — 0001	NE — 0110
GE — 1100	PL — 1010
GT — 1110	T — 0000
HI — 0010	VC — 1000
LE — 1111	VS — 1001

実効アドレス・フィールド——ディスティネーション・オペランドのアドレッシング・モードを指定．指定アドレスのバイトはすべて1またはすべて0に書きかわる．データと可変のアドレッシング・モードのみ可．

MC68,000 — Scc INSTRUCTION EXAMPLE

```
*    SCC INSTRUCTION EXAMPLE
*
*THE SCC INSTRUCTION ALLOWS THE USER
*TO REMEMBER THE RESULT OF AN INSTRUCTION
*AND ACT UPON IT AT SOME LATER TIME.
*
 ORG $2000
 CLR D0
 CLR D1
 CMP D0,D1
 SEQ D2
 ADDQ #1,D0
 CMP D0,D1
 SEQ D2
 END
```

STOP (Load Status Register and Stop SRロードとストップ)

オペレーション：イミディエイト・データ→SR; STOP

STOP ♯×××

アセンブラ：STOP ♯×××

属性：サイズなし

説明：イミディエイト・オペランドがSR全体に移され，PCは次の命令を示すよう進められ，プロセッサは命令フェッチと実行をストップします．トレース，割込みまたはリセットのエクセプション発生により実行が再開されます．STOP命令実行時にTビットが1ならエクセプション・トレースが発生します．もし現在のプロセッサの優先順位より高い優先度の割込み要求が来ると，割込みエクセプションが発生しますが，そうでなければ割込み要求は無効です．外部リセットを行うとエクセプション・リセット・シーケンスを開始させます．

（注） これは特権命令です．

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

N——イミディエイト・オペランドに従ってセット．

Z——イミディエイト・オペランドに従ってセット．

V——イミディエイト・オペランドに従ってセット．

C——イミディエイト・オペランドに従ってセット．

X——イミディエイト・オペランドに従ってセット．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0
イミディエイト・データ															

命令フィールド：

イミディエイト・フィールド——SRにロードするイミディエイト・データを指定．

SUB (Subtract Binary 2進減算)

オペレーション：(ディスティネーション)—(ソース)→ディスティネーション

SUB <ea> FROM Dn

SUB Dn FROM <ea>

アセンブラ：SUB <ea>, Dn

SUB Dn, <ea>

属性：サイズ=(バイト，ワード，ロングワード)

説明：ソース・オペランドをディスティネーション・オペランドから減算し，結果をディスティネーションにストアします．オペレーションのサイズは，バイト，ワード，ロングワードに指定できます．命令の中のモードでどちらのオペランドがソースで，どちらのオペランドがディスティネーションかを，およびそのオペレーション・サイズを示します．

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

N——結果が負であればセット，それ以外のときクリア．

Z——結果が0あればセット，それ以外のときクリア．

V——オーバーフローが発生すればセット，それ以外のときクリア．

C——キャリが発生すればセット，それ以外のときクリア．

X——キャリ・ビットと同様．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	レジスタ			Op-モード			実効アドレス					

命令フィールド：

レジスタ・フィールド——データ・レジスタの番号を指定．

Op-モード・フィールド——

バイト	ワード	ロングワード	オペレーション
000	001	010	(<Dn>)—(<ea>) →<Dn>
100	101	110	(<ea>) —(<Dn>)→<ea>

実効アドレス・フィールド——アドレッシング・モードを指定．

もし指定された位置がソース・オペランドなら，すべてのアドレッシング・モードが可．

もしオペランド・サイズがバイトなら，アドレス・レジスタ直接アドレッシング・モードは不可．

もし指定された位置がディスティネーション・オペランドなら，可変とメモリのアドレッシング・モードのみ可．

SUBA (Subtract Address アドレス減算)

オペレーション：(ディスティネーション)−(ソース)→ディスティネーション
SUB <ea> FROM An

アセンブラ：SUBA <ea>, An

属性：サイズ＝(ワード，ロングワード)

説明：ソース・オペランドをディスティネーション・アドレス・レジスタから減算し，結果をアドレス・レジスタにストアします．オペレーションのサイズはワードかロングワードに指定できます．

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化せず．
Z——変化せず．
V——変化せず．
C——変化せず．
X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	レジスタ			Op-モード			実効アドレス					

命令フィールド：

レジスタ・フィールド——アドレス・レジスタ番号を指定，これはいつもディスティネーションとなる．

Op-モード・フィールド——オペレーション・サイズを指定．

011——ワード・オペレーション．ソース・オペランドがロングワードに符号拡張され，アドレス・レジスタ全32ビットがオペレーションされます．

111——ロングワード・オペレーション

実効アドレス・フィールド——ソース・オペランドのアドレッシング・モードを指定．すべてのアドレッシング・モードが可．

SUBI (Subtract Immediate イミディエイト減算)

オペレーション：(ディスティネーション)—イミディエイト・データ→ディスティネーション
SUB ♯<データ> FROM <ea>

アセンブラ：SUBI ♯<データ> <ea>

属性：サイズ＝(バイト，ワード，ロングワード)

説明：イミディエイト・データをディスティネーション・オペランドから減算し，結果をディスティネーション位置にストアします．オペレーションのサイズは，バイト，ワード，ロングワードに指定できます．イミディエイト・データのサイズはオペレーション・サイズと合致します．

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

N——結果が負であればセット，それ以外のときクリア．
Z——結果が0であればセット，それ以外のときクリア．
V——オーバフローが発生すればセット，それ以外のときクリア．
C——キャリが発生すればセット，それ以外のときクリア．
X——キャリ・ビットと同じ．

命令フォーマット：

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td colspan="2">サイズ</td><td colspan="6">実効アドレス</td></tr>
<tr><td colspan="8">ワード・データ(16ビット)</td><td colspan="8">バイト・データ(8ビット)</td></tr>
<tr><td colspan="16">ロング・データ(32ビット、先行ワードを含む)</td></tr>
</table>

命令フィールド：

サイズ・フィールド——オペレーション・サイズを指定．
00——バイト・オペレーション
01——ワード・オペレーション
10——ロングワード・オペレーション

実効アドレス・フィールド——ディスティネーション・オペランドのアドレッシング・モードを指定．データと可変のアドレッシング・モードのみ可

イミディエイト・フィールド——(命令の後のデータが入ります)
サイズ＝00，データはイミディエイト・ワードの下位バイト（8ビット）
サイズ＝01，データはイミディエイト・ワード（16ビット）

サイズ=10，データは次の2つのイミディエイト・ワード（32ビット）

SUBQ (Subtract Quick　クイック減算)

オペレーション：(ディスティネーション)—イミディエイト・データ→ディスティネーション
SUBQ ＃<データ> FROM <ea>

アセンブラ：SUBQ ＃<データ> <ea>

属性：サイズ＝(バイト，ワード，ロングワード)

説明：イミディエイト・データをディスティネーション・オペランドから減算します．データ範囲は1-8です．オペレーション・サイズはバイト，ワード，ロングワードに指定できます．アドレス・レジスタでのワードおよびロングワードオペレーションも許されます．

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

N——結果が負であればセット，それ以外のときクリア．
Z——結果が0であればセット，それ以外のときクリア．
V——オーバーフローが発生すればセット，それ以外のときクリア．
C——キャリが発生すればセット，それ以外のときクリア．
X——キャリ・ビットと同じ．
アドレス・レジスタからの減算がなされる場合，コンディション・コードは変化しません．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	データ			1	サイズ		実効アドレス					

命令フィールド：

データ・フィールド——3ビットのイミディエイト・データであり，0は8を1-7は1～7を表わす．
サイズ・フィールド——オペレーション・サイズを指定．
00——バイト・オペレーション
01——ワード・オペレーション
10——ロングワード・オペレーション
実効アドレス・フィールド——ディスティネーション・オペランドのアドレッシング・モードを指定．可変アドレッシング・モードのみ可．ただし，オペレーション・サイズがバイトならアドレス・レジスタ直接アドレッシング・モードは不可

SUBX (Subtract with Extend 拡張付き減算)

オペレーション：(ディスティネーション)—(ソース)—X→ディスティネーション
SUBX Dy FROM Dx
SUBX Ay@—FROM Ax@—

アセンブラ：SUBX Dy, Dx
SUBX—(Ay), —(Ax)

属性：サイズ＝(バイト，ワード，ロングワード)

説明：ソース・オペランドと拡張ビットをディスティネーション・オペランドから減算し，結果をディスティネーションの位置にストアします．オペランドは2つの方法でアドレスできます．

1 データ・レジスタからデータ・レジスタ： オペランドは命令で指定されるデータ・レジスタ内にある．

2 メモリからメモリ： オペランドはメモリ内にあり，命令で指定されるアドレス・レジスタを用いたプリデクリメント・アドレッシング・モードでアドレスされます．

オペレーションのサイズは，バイト，ワード，ロングワードに指定できます．

コンディション・コード：

X	N	Z	V	C
*	*	*	*	*

N——結果が負であればセット．それ以外のときクリア．
Z——結果が0であればセット．それ以外のときクリア．
V——オーバフローが発生すればセット．それ以外のときクリア．
C——キャリが発生すればセット．それ以外のときクリア．
X——キャリ・ビットと同じ．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	レジスタ Rx			1	サイズ		0	0	R/M	レジスタ Ry		

命令フィールド：

レジスタ Rx フィールド——ディスティネーション・レジスタの番号を指定．
R/M=0，データ・レジスタの番号．
R/M=1，アドレス・レジスタの番号（プリデクリメント・アドレス・モードで使用）．

サイズ・フィールド——オペレーション・サイズを指定．
00——バイト・オペレーション
01——ワード・オペレーション
10——ロングワード・オペレーション

R/M フィールド——オペランドのアドレッシング・モードを指定．

0——データ・レジスタからデータ・レジスタのオペレーション.

1——メモリからメモリのオペレーション.

レジスタ Ry フィールド——ソース・レジスタの番号を指定.

R/M=0, データ・レジスタの番号.

R/M=1, アドレス・レジスタ（プリデクリメント・アドレス・モードで使用）の番号.

SWAP (Swap Register Halves　レジスタ半分スワップ)

オペレーション： レジスタ〔31：16〕↔レジスタ〔15：0〕

SWAP Dn

アセンブラ： SWAP Dn

属性： サイズ＝(ワード)

説明： データ・レジスタの上下半分（各16ビット）の内容を入れかえます．

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

N——32ビットの結果の最上位ビットが1であればセット．それ以外のときクリア．

Z——32ビットの結果が0であればセット．それ以外のときクリア．

V——常にクリア．

C——常にクリア．

X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	0	レジスタ		

命令フィールド：

レジスタ・フィールド——データ・レジスタの番号を指定．

TAS (Test and Set an Operand　オペランド・テストとセット)

オペレーション：(ディスティネーション)Tesfed→CC; 1→♯7 OF ディスティネーション

TAS <ea>

アセンブラ：TAS <ea>

属性：サイズ＝(バイト)

説明：まず指定されたバイト・オペランドをテストし，その結果に従ってコンディション・コードのNとZをセットします．その後オペランドの最上位ビットを“1”にします．

オペレーションは，リードーモディファイーライト・サイクルを使用しており複数プロセッサの同期のためにバス・サイクルの分割は不可能になっています．

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

N——オペランドの最上位ビットが1であればセット，それ以外のときクリア．

Z——オペランドが0であればセット，それ以外のときはクリア．

V——常にクリア．

C——常にクリア．

X——変化せず．

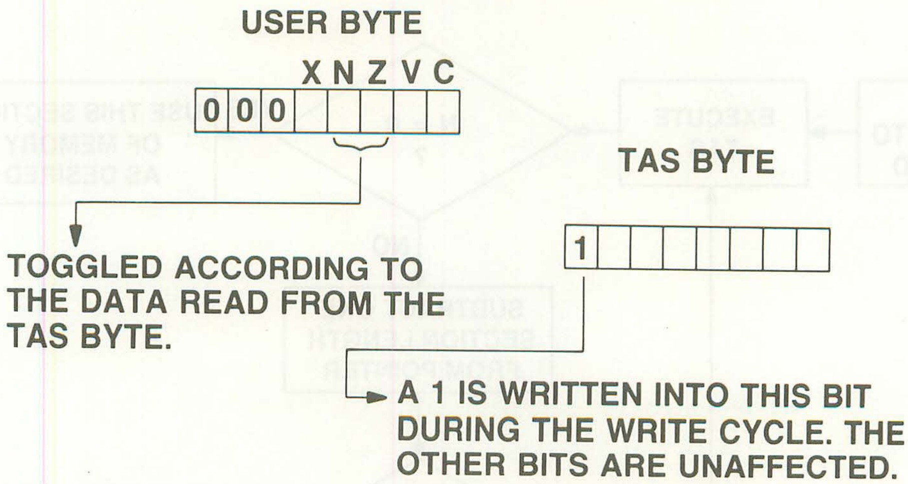
命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	実効アドレス					

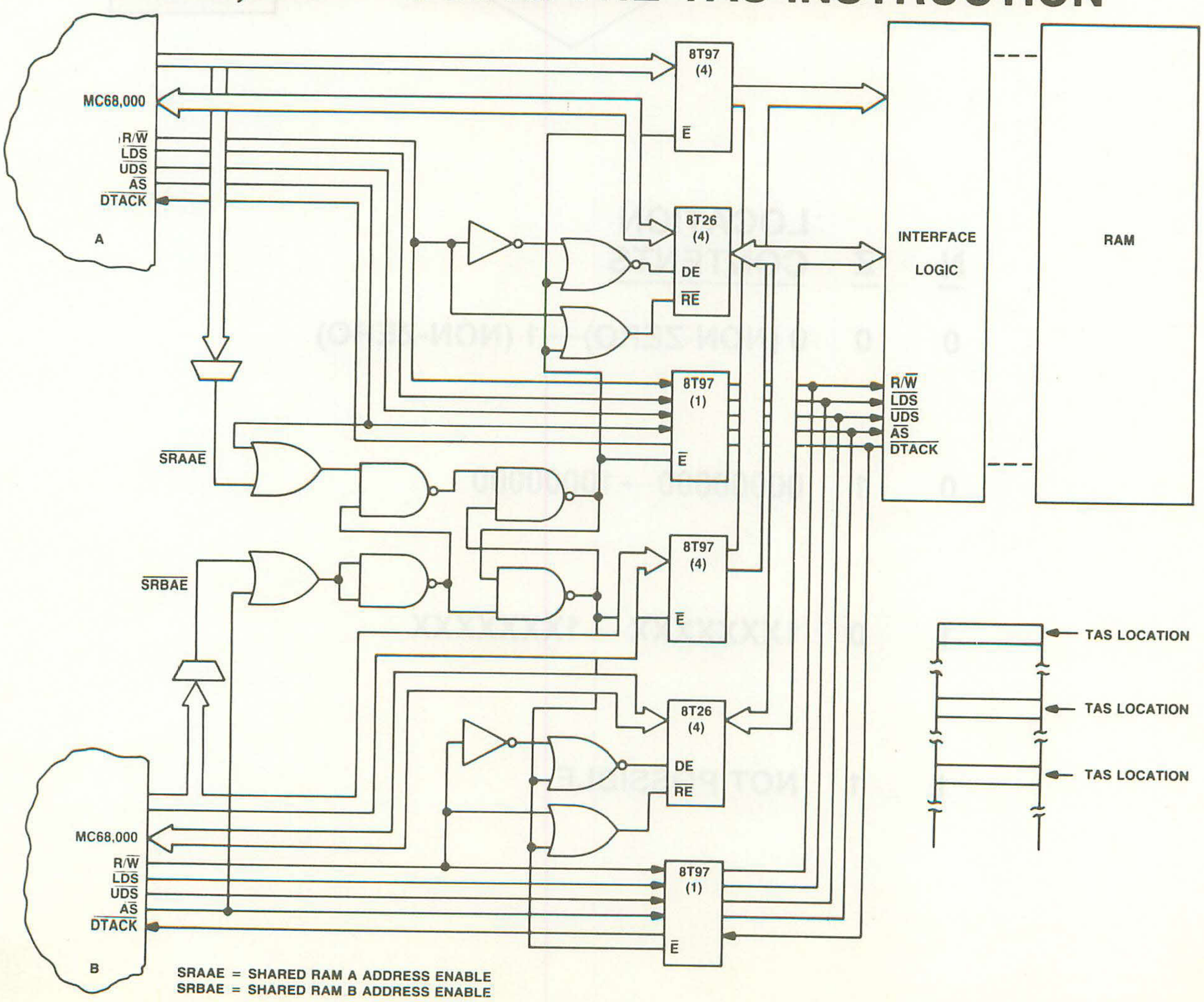
命令フィールド：

実効アドレス・フィールド——テストされるバイトのアドレッシング・モードを指定．データと可変のアドレッシング・モードのみ可

MC68,000 — USING THE TAS INSTRUCTION

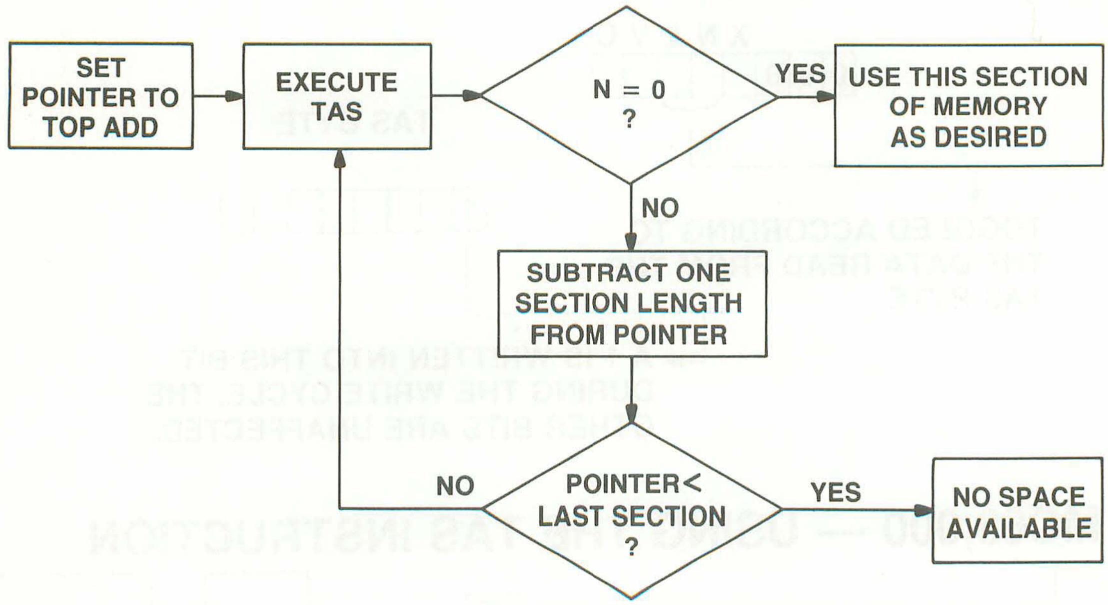


MC68,000 — USING THE TAS INSTRUCTION



MC68,000 — USING THE TAS INSTRUCTION

ALLOCATING MEMORY



N	Z	LOCATION CONTENTS
0	0	0 (NON-ZERO) → 1 (NON-ZERO)
0	1	00000000 → 10000000
1	0	1XXXXXXX → 1XXXXXXX
1	1	NOT POSSIBLE

TRAP (Trap トラップ)

オペレーション：PC → SSP@—; SR → SSP@—; (ベクタ) → PC
TRAP ♯<ベクタ>

アセンブラ：TRAP ♯<ベクタ>

属性：サイズなし

説明：プロセッサはエクセプション・プロセッシングを開始します．命令の下位4ビットで指定される TRAP 命令のエクセプション・ベクタを参照するよう，ベクタ番号が作られます．16個の TRAP ベクタが使用できます．

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化せず．
Z——変化せず．
V——変化せず．
C——変化せず．
X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	0	ベクタ			

命令フィールド：

ベクタ・フィールド——TRAP のベクタ番号を指定．

TRAPV (Trap on Overflow　オーバフロー・トラップ)

オペレーション： If V then TRAP
TRAPV

アセンブラ： TRAPV

属性： サイズなし

説明： オーバフロー・コンディションがあればプロセッサはエクセプション・プロセッシングを開始します．TRAPV エクセプション・ベクタを参照するようベクタ番号が作られます．オーバフロー・コンディションがなければ何もせず実行は次の命令から続けられます．

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化せず．
Z——変化せず．
V——変化せず．
C——変化せず．
X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0

TST (Test an Operand オペランド・テスト)

オペレーション：（ディスティネーション）Tested→CC
TST <ea>

アセンブラ：TST <ea>

属性：サイズ＝(バイト，ワード，ロングワード)

説明：ディスティネーション・オペランドを0と比較します．テスト結果に応じてコンディション・コードがセットされるだけで，オペランドの内容は変化しません．オペレーションのサイズはバイト，ワード，ロングワードに指定できます．

コンディション・コード：

X	N	Z	V	C
—	*	*	0	0

N——結果が負であればセット，それ以外のときクリア．
Z——結果が0であればセット，それ以外のときクリア．
V——常にクリア
C——常にヤリア
X——変化せず

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	サイズ		実効アドレス					

命令フィールド：

サイズ・フィールド——オペレーション・サイズを指定．
00——バイト・オペレーション
01——ワード・オペレーション
10——ロングワード・オペレーション
実効アドレス・フィールド——ディスティネーション・オペランドのアドレッシング・モードを指定．アドレッシング・モードのみ可．

UNLK (Unlink　リンク解除)

オペレーション：An → SP; SP@+→ An
UNLK An

アセンブラ：UNLK An

属性：サイズなし

説明：指定されたアドレス・レジスタがスタック・ポインタにロードされます．つぎにアドレス・レジスタにスタックの先頭からとり出されたロングワードがロードされ，その後スタック・ポインタはインクリメントされます．

コンディション・コード：

X	N	Z	V	C
—	—	—	—	—

N——変化せず．
Z——変化せず．
V——変化せず．
C——変化せず．
X——変化せず．

命令フォーマット：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	1	レジスタ		

命令フィールド：

レジスタ・フィールド——アドレス・レジスタの番号を指定．

付録C　命令実行時間

本付録では，外部クロック（CLK：external clock）周波数（サイクル）に関連する命令実行時間について述べます．このタイミング・データにおいてメモリのサイクル・タイムはプロセッサの外部クロック入力の4周期より短いと仮定しています．つまり，バス・サイクルにおいて待ち状態に入ることがない場合です．各命令のバス・リード・サイクル，およびライト・サイクル数もまたタイミング・データに含まれており，実行周期数に続くカッコで（r/w）のように示します．ここで，rはリード・サイクル数，wはライト・サイクル数を示します．

（注）命令実行サイクルには，命令のフェッチおよび対応するすべてのオペランドのフェッチおよびストアが含まれます．

C.1　実効アドレス・オペランド計算時間

表C-1は,命令の実効アドレスの計算に必要なクロック周期数を示します．これにはすべての拡張語，アドレス計算，およびメモリ・オペランドのフェッチが含まれます．バス・リード／ライト・サイクル数は（r/w）のようにカッコで示します．実効アドレスの処理には，ライト・サイクルは含まれないことに注意してください．

表C-1　実効アドレス計算の時間

Addressing Mode		Byte, Word	Long
	Register		
Dn	Data Register Direct	0(0/0)	0(0/0)
An	Address Register Direct	0(0/0)	0(0/0)
	Memory		
An@	Address Register Indirect	4(1/0)	8(2/0)
An@+	Address Register Indirect with Postincrement	4(1/0)	8(2/0)
An@−	Address Register Indirect with Predecrement	6(1/0)	10(2/0)
An@(d)	Address Register Indirect with Displacement	8(2/0)	12(3/0)
An@(d, ix)*	Address Register Indirect with Index	10(2/0)	14(3/0)
xxx.W	Absolute Short	8(2/0)	12(3/0)
xxx.L	Absolute Long	12(3/0)	16(4/0)
PC@(d)	Program Counter with Displacement	8(2/0)	12(3/0)
PC@(d, ix)*	Program Counter with Index	10(2/0)	14(3/0)
#xxx	Immediate	4(1/0)	8(2/0)

＊インデックス・レジスタ（ix）のサイズは，実行時間に影響を及ぼさない．

C.2　MOVE 命令クロック周期

表C-2，3はMOVE命令に要するクロック周期数を示します．このデータには命令のフェッチ，オペランド・リード，およびオペランド・ライトが含まれます．バス・リード／ライト・サイクル数を（r/w）のようにカッコで示します．

表C-2　バイトおよびワード命令の MOVE クロック周期

Source	Destination								
	Dn	An	An@	An@+	An@−	An@(d)	An@(d, ix)	xxx.W	xxx.L
Dn	4(1/0)	4(1/0)	9(1/1)	9(1/1)	9(1/1)	13(2/1)	15(2/1)	13(2/1)	17(3/1)
An	4(1/0)	4(1/0)	9(1/1)	9(1/1)	9(1/1)	13(2/1)	15(2/1)	13(2/1)	17(3/1)
An@	8(2/0)	8(2/0)	13(2/1)	13(2/1)	13(2/1)	17(3/1)	19(3/1)	17(3/1)	21(4/1)
An@+	8(2/0)	8(2/0)	13(2/1)	13(2/1)	13(2/1)	17(3/1)	19(3/1)	17(3/1)	21(4/1)
An@−	10(2/0)	10(2/0)	15(2/1)	15(2/1)	15(3/1)	19(3/1)	21(3/1)	19(3/1)	23(4/1)
An@(d)	12(3/0)	12(3/0)	17(3/1)	17(3/1)	17(3/1)	21(4/1)	23(4/1)	21(4/1)	15(5/1)
An@(d, ix)*	14(3/0)	14(3/0)	19(3/1)	19(3/1)	19(3/1)	23(4/1)	25(4/1)	23(4/1)	27(5/1)
xxx.W	12(3/0)	12(3/0)	17(3/1)	17(3/1)	17(3/1)	21(4/1)	23(4/1)	21(4/1)	25(5/1)
xxx.L	16(4/0)	16(4/0)	21(4/1)	21(4/1)	21(4/1)	25(5/1)	27(5/1)	25(5/1)	29(6/1)
PC@(d)	12(3/0)	12(3/0)	17(3/1)	17(3/1)	17(3/1)	21(4/1)	23(4/1)	21(4/1)	25(5/1)
PC@(d, ix)*	14(3/0)	14(3/0)	19(3/1)	19(3/1)	19(3/1)	23(4/1)	25(4/1)	23(4/1)	27(5/1)
#xxx	8(2/0)	8(2/0)	13(2/1)	13(2/1)	13(2/1)	17(3/1)	19(3/1)	17(3/1)	21(4/1)

＊インデックス・レジスタ（ix）のサイズは，実行時間に影響を及ぼさない．

表C-3　ロングワード命令の MOVE クロック周期

Source	Destination								
	Dn	An	An@	An@+	An@−	An@(d)	An@(d, ix)	xxx.W	xxx.L
Dn	4(1/0)	4(1/0)	14(1/2)	14(1/2)	16(1/2)	18(2/2)	20(2/2)	18(2/2)	22(3/2)
An	4(1/0)	4(1/0)	14(1/2)	14(1/2)	16(1/2)	18(2/2)	20(2/2)	18(2/2)	22(3/2)
An@	12(3/0)	12(3/0)	22(3/2)	22(3/2)	22(3/2)	26(4/2)	28(4/2)	26(4/2)	30(5/2)
An@+	12(3/0)	12(3/0)	22(3/2)	22(3/2)	22(3/2)	26(4/2)	28(4/2)	26(4/2)	30(5/2)
An@−	14(3/0)	14(3/0)	24(3/2)	24(3/2)	24(3/2)	28(4/2)	30(4/2)	28(4/2)	32(5/2)
An@(d)	16(4/0)	16(4/0)	26(4/2)	26(4/2)	26(4/2)	30(5/2)	35(5/2)	30(5/2)	34(6/2)
An@(d, ix)*	18(4/0)	18(4/0)	28(4/2)	28(4/2)	28(4/2)	32(5/2)	34(5/2)	32(5/2)	36(6/2)
xxx.W	16(4/0)	16(4/0)	26(4/2)	26(4/2)	26(4/2)	30(5/2)	32(5/2)	30(5/2)	34(6/2)
xxx.L	20(5/0)	20(5/0)	30(5/2)	30(5/2)	30(5/2)	34(6/2)	36(6/2)	34(6/2)	38(7/2)
PC@(d)	16(4/0)	16(4/0)	26(4/2)	26(4/2)	26(4/2)	30(5/2)	32(5/2)	30(5/2)	34(6/2)
PC@(d, ix)*	18(4/0)	18(4/0)	28(4/2)	28(4/2)	28(4/2)	32(5/2)	34(5/2)	32(5/2)	36(6/2)
#xxx	12(3/0)	12(3/0)	22(3/2)	22(3/2)	22(3/2)	26(4/2)	28(4/2)	26(4/2)	30(5/2)

＊インデックス・レジスタ（ix）のサイズは，実行時間に影響を及ぼさない．

C.3　標準命令クロック周期

表C-4に示すクロック周期数は，オペレーションの実行，結果のストアそして次の命令の読み出しに要する時間を示します．バス・リード／ライト・サイクルの数を（r/w）のようにカッコで示します．クロック周期数とリード／ライト・サイクル数を加えることは，それらの実効アドレスを加えることを意味しています．

表C-4の項目は，次のような意味を持っています：An＝アドレス・レジスタ・オペランド，Dn＝データ・レジスタ・オペランド，ea＝実効アドレスによって指定されるオペランド，M＝メモリ実効アドレス・オペランドです．

表C-4 標準命令クロック周期

Instruction	Size	op <ea>, An	op <ea>, Dn	op Dn, <M>
ADD	Byte, Word	8(1/0)+	4(1/0)+	9(1/1)+
	Long	6(1/0)+**	6(1/0)+**	14(1/2)+
AND	Byte, Word	—	4(1/0)+	9(1/1)+
	Long	—	6(1/0)+**	14(1/2)+
CMP	Byte, Word	6(1/0)+	4(1/0)+	—
	Long	6(1/0)+	6(1/0)+	—
DIVS	—	—	158(1/0)+*	—
DIVU	—	—	140(1/0)+*	—
EOR	Byte, Word	—	4(1/0)***	9(1/1)+
	Long	—	8(1/0)***	14(1/2)+
MULS	—	—	70(1/0)+*	—
MULU	—	—	70(1/0)+*	—
OR	Byte, Word	—	4(1/0)+	9(1/1)+
	Long	—	6(1/0)+**	14(1/2)+
SUB	Byte, Word	8(1/0)+	4(1/0)+	9(1/1)+
	Long	6(1/0)+**	6(1/0)+**	14(1/2)+

† 実効アドレス計算時間を加算.
* 最大値を示す.
** 実効アドレスがレジスタ直接の場合，命令は合計8クロック周期.
*** 使用できる実効アドレス・モードは，データ・レジスタ直接だけ.

C.4 イミディエイト命令クロック周期

表C-5に示すクロック周期数は，イミディエイト・オペランドのフェッチ，オペレーションの実行結果をストア，そして次の命令を読み出す時間を示します．バス・リード／ライト・サイクル数を（r/w）のように示します．クロック周期数とリード／ライト・サイクル数を加えるということは，それらの実効アドレスを加算することを意味します．

表C-5の項目は，次のような意味を持っています：#=イミディエイト・オペランド，Dn=データ・レジスタ・オペランド，M=メモリ・オペランド，SR=ステータス・レジスタ．

表C-5 イミディエイト命令クロック周期

Instruction	Size	op #, Dn	op #, M	op #, SR
ADDI	Byte, Word	8(2/0)	13(2/1)+	—
	Long	16(3/0)	22(3/2)+	—
ADDQ	Byte, Word	4(1/0)	9(1/1)+	—
	Long	8(1/0)	14(1/2)+	—
ANDI	Byte, Word	8(2/0)	13(2/1)+	20(3/0)
	Long	16(3/0)	22(3/2)+	—
CMPI	Byte, Word	8(2/0)	8(2/0)+	—
	Long	14(3/0)	12(3/0)+	—
EORI	Byte, Word	8(2/0)	13(2/1)+	20(3/0)
	Long	16(3/0)	22(3/2)+	—
MOVEQ	Long	4(1/0)	—	—
ORI	Byte, Word	8(2/0)	13(2/1)+	20(3/0)
	Long	16(3/0)	22(3/2)+	—
SUBI	Byte, Word	8(2/0)	13(2/1)+	—
	Long	16(3/0)	22(3/2)+	—
SUBQ	Byte, Word	4(1/0)	9(1/1)+	—
	Long	8(1/0)	14(1/2)+	—

+ 実効アドレス計算時間を加算.

C.5 シングル・オペランド命令クロック周期

表C-6は，シングル・オペランド命令に必要なクロック周期数を示します．バス・リード／ライト・サイクル数を（r/w）のように示します．クロック周期数とリード／ライト・サイクル数を加えるということは，それらの実効アドレスを加算することを意味します．

表C-6 シングル・オペランド命令クロック周期

Instruction	Size	Register	Memory
CLR	Byte, Word	4(1/0)	9(1/1)+
	Long	6(1/0)	14(1/2)+
NBCD	Byte	6(1/0)	9(1/1)+
NEG	Byte, Word	4(1/0)	9(1/1)+
	Long	6(1/0)	14(1/2)+
NEGX	Byte, Word	4(1/0)	9(1/1)+
	Long	6(1/0)	14(1/2)+
NOT	Byte, Word	4(1/0)	9(1/1)+
	Long	6(1/0)	14(1/2)+
Scc	Byte, False	4(1/0)	9(1/1)+
	Byte, True	6(1/0)	9(1/1)+
TAS	Byte	4(1/0)	11(1/1)+
TST	Byte, Word	4(1/0)	4(1/0)+
	Long	4(1/0)	4(1/0)+

+ 実効アドレス計算時間を加算.

C.6 シフト／ローテイト命令クロック周期

表 C-7は，シフト／ローテイトに必要なクロック周期数を示します．バス・リード／ライト・サイクル数を（r/w）のようにカッコで示します．クロック周期数とリード／ライト・サイクル数を加えることは，それらの実効アドレスを加算することを意味します．

表C-7 シフト／ローテイト命令クロック周期

Instruction	Size	Register	Memory
ASR, ASL	Byte, Word	6 + 2n(1/0)	9(1/1)+
	Long	8 + 2n(1/0)	—
LSR, LSL	Byte, Word	6 + 2n(1/0)	9(1/1)+
	Long	8 + 2n(1/0)	—
ROR, ROL	Byte, Word	6 + 2n(1/0)	9(1/1)+
	Long	8 + 2n(1/0)	—
ROXR, ROXL	Byte, Word	6 + 2n(1/0)	9(1/1)+
	Long	8 + 2n(1/0)	—

＋ 実効アドレス計算時間を加算．
nはシフト回数．

C.7 ビット処理命令クロック周期

表 C-8は，ビット処理命令に要するクロック周期数を示します．バス・リード／ライト・サイクル数を（r/w）のように示します．クロック周期数とリード／ライト・サイクル数を加えることは，それらの実効アドレスを加算することを意味します．

表C-8 ビット処理命令クロック周期

Instruction	Size	Dynamic		Static	
		Register	Memory	Register	Memory
BCHG	Byte	—	9(1/1)+	—	13(2/1)+
	Long	8(1/0)*	—	12(2/0)*	—
BCLR	Byte	—	9(1/1)+	—	13(2/1)+
	Long	10(1/0)*	—	14(2/0)*	—
BSET	Byte	—	9(1/1)+	—	13(2/1)+
	Long	8(1/0)*	—	12(2/0)*	—
BTST	Byte	—	4(1/0)+	—	8(2/0)+
	Long	6(1/0)	—	10(2/0)	—

＋ 実効アドレス計算時間を加算．
＊ 最大値を示す．

C.8 コンディション付き命令クロック周期

表C-9は，コンディション付き命令に要するクロック周期数を示します．バス・リード／ライト・サイクル数を(r/w) のように示します．クロック周期数とリード／ライト・サイクル数を加えることは，それらの実効アドレスを加算することを意味します．

表C-9 コンディション付き命令クロック期周

Instruction	Displacement	Trap or Branch Taken	Trap or Branch Not Taken
B_{CC}	Byte	10(1/0)	8(1/0)
	Word	10(1/0)	12(2/0)
BRA	Byte	10(1/0)	—
	Word	10(1/0)	—
BSR	Byte	20(2/2)	—
	Word	20(2/2)	—
DB_{CC}	cc true	—	12(2/0)
	cc false	10(2/0)	14(3/0)
CHK	—	43(5/3)+*	8(1/0)+
TRAP	—	37(4/3)	—
TRAPV	—	37(5/3)	4(1/0)

＋ 実効アドレス計算時間を加算.

＊ 最大値を示す.

C.9 JMP, JSR, LEA, PEA, MOVEM 命令クロック周期

表 C-10 JMP, JSR, LEA, PEA, MOVE 命令クロック周期

Instr	Size	An@	An@+	An@−	An@(d)	An@(d, ix)*	xxx.W	xxx.L	PC@(d)	PC@(d, ix)*
JMP	—	8(2/0)	—	—	10(2/0)	14(3/0)	10(2/0)	12(3/0)	10(2/0)	14(3/0)
JSR	—	18(2/2)	—	—	20(2/2)	24(2/2)	20(2/2)	22(3/2)	20(2/2)	24(2/2)
LEA	—	4(1/0)	—	—	8(2/0)	12(2/0)	8(2/0)	12(3/0)	8(2/0)	12(2/0)
PEA	—	14(1/2)	—	—	18(2/2)	22(2/2)	18(2/2)	22(3/2)	18(2/2)	22(2/2)
MOVEM	Word	12+4n (3+n/0)	12+4n (3+n/0)	— —	16+4n (4+n/0)	18+4n (4+n/0)	16+4n (4+n/0)	20+4n (5+n/0)	16+4n (4+n/0)	18+4n (4+n/0)
M → R	Long	12+8n (3+2n/0)	12+8n (3+2n/0)	— —	16+8n (4+2n/0)	18+8n (4+2n/0)	16+8n (4+2n/0)	20+8n (5+2n/0)	16+8n (4+2n/0)	18+8n (4+2n/0)
MOVEM	Word	8+5n (2/n)	— —	8+5n (2/n)	12+5n (3/n)	14+5n (3/n)	12+5n (3/n)	16+5n (4/n)	— —	— —
R → M	Long	8+10n (2/2n)	— —	8+10n (2/2n)	12+10n (3/2n)	14+10n (3/2n)	12+10n (3/2n)	16+10n (4/2n)	— —	— —

n：MOVE するレジスタ数

＊：インデックス・レジスタ (ix) のサイズは命令の実行時間に影響を及ぼさない．

表 C-10は，ジャンプ（JMP: jump），ジャンプ・ツー・サブルーチン（JSR: jump to subroutine），ロード実効アドレス（LEA: load effective address），プッシュ実効アドレス（PEA: push effective address），ムーブ多重レジスタ命令（MOVEM: move multiple registers instructions）に要するクロックの周期数を示します．バス・リード／ライト・サイクル数を（r/w）のように示します．

C.10 多重精度命令クロック周期

表 C-11は，多重精度命令に要するクロック周期数を示します．クロック周期数は，2つのオペランドのフェッチ，操作の実行，結果のストアそして次の命令を読み出すまでの時間を示します．リード／ライト・サイクル数を（r/w）のように示します．

表 C-11の項目は，次のような意味をもっています：Dn＝データ・レジスタ・オペランド，M＝メモリ・オペランド．

表C-11 多重精度命令クロック周期

Instruction	Size	op Dn, Dn	op M, M
ADDX	Byte, Word	4(1/0)	19(3/1)
	Long	8(1/0)	32(5/2)
CMPM	Byte, Word	—	12(3/0)
	Long	—	20(5/0)
SUBX	Byte, Word	4(1/0)	19(3/1)
	Long	8(1/0)	32(5/2)
ABCD	Byte	6(1/0)	19(3/1)
SBCD	Byte	6(1/0)	19(3/1)

C.11 その他の命令クロック周期

表 C-12は，その他の命令に要するクロック周期数を示しています．バス・リード／ライト・サイクル数を(r/w)のようにカッコで示します．クロック周期数とリード／ライト・サイクル数を加えることは，それらの実効アドレスを加算することを意味します．

表C-12 その他の命令クロック周期

Instruction	Size	Register	Memory	Register→Memory	Memory→Register
MOVE from SR	—	6(1/0)	9(1/1)+	—	—
MOVE to CCR	—	12(2/0)	12(2/0)+	—	—
MOVE to SR	—	12(2/0)	12(2/0)+	—	—
MOVEP	Word	—	—	18(2/2)	16(4/0)
	Long	—	—	28(2/4)	24(6/0)
EXG	—	6(1/0)	—	—	—
EXT	Word	4(1/0)	—	—	—
	Long	4(1/0)	—	—	—
LINK	—	18(2/2)	—	—	—
MOVE from USP	—	4(1/0)	—	—	—
MOVE to USP	—	4(1/0)	—	—	—
NOP	—	4(1/0)	—	—	—
RESET	—	132(1/0)	—	—	—
RTE	—	20(5/0)	—	—	—
RTR	—	20(5/0)	—	—	—
RTS	—	16(4/0)	—	—	—
STOP	—	4(0/0)	—	—	—
SWAP	—	4(1/0)	—	—	—
UNLK	—	12(3/0)	—	—	—

+ 実効アドレス計算時間を加算

C.12 エクセプション・プロセッシング・クロック期間

表 C-13は，エクセプション・プロセッシングに要するクロックの周期数を示します．クロックの周期数はすべてのスタック処理，ベクタ・フェッチ，ハンドラ・ルーチンの等1命令のフェッチに要する時間を示します．バス・リード／ライト・サイクル数は（r/w）のように示します．クロック周期数とリード／ライト・サイクル数を加えるということは，それらの実効アドレスを加算することを意味しています．

表C-13 エクセプション・プロセッシング・クロック周期

Exception	Periods
Address Error	57(4/7)
Bus Error	57(4/7)
Interrupt	47(5/3)*
Illegal Instruction	37(4/3)
Privileged Instruction	37(4/3)
Trace	37(4/3)

* 割込み・アクノレッジのバス・サイクルは4クロック周期とする

THE M68000 FAMILY — AN INVESTMENT IN THE FUTURE

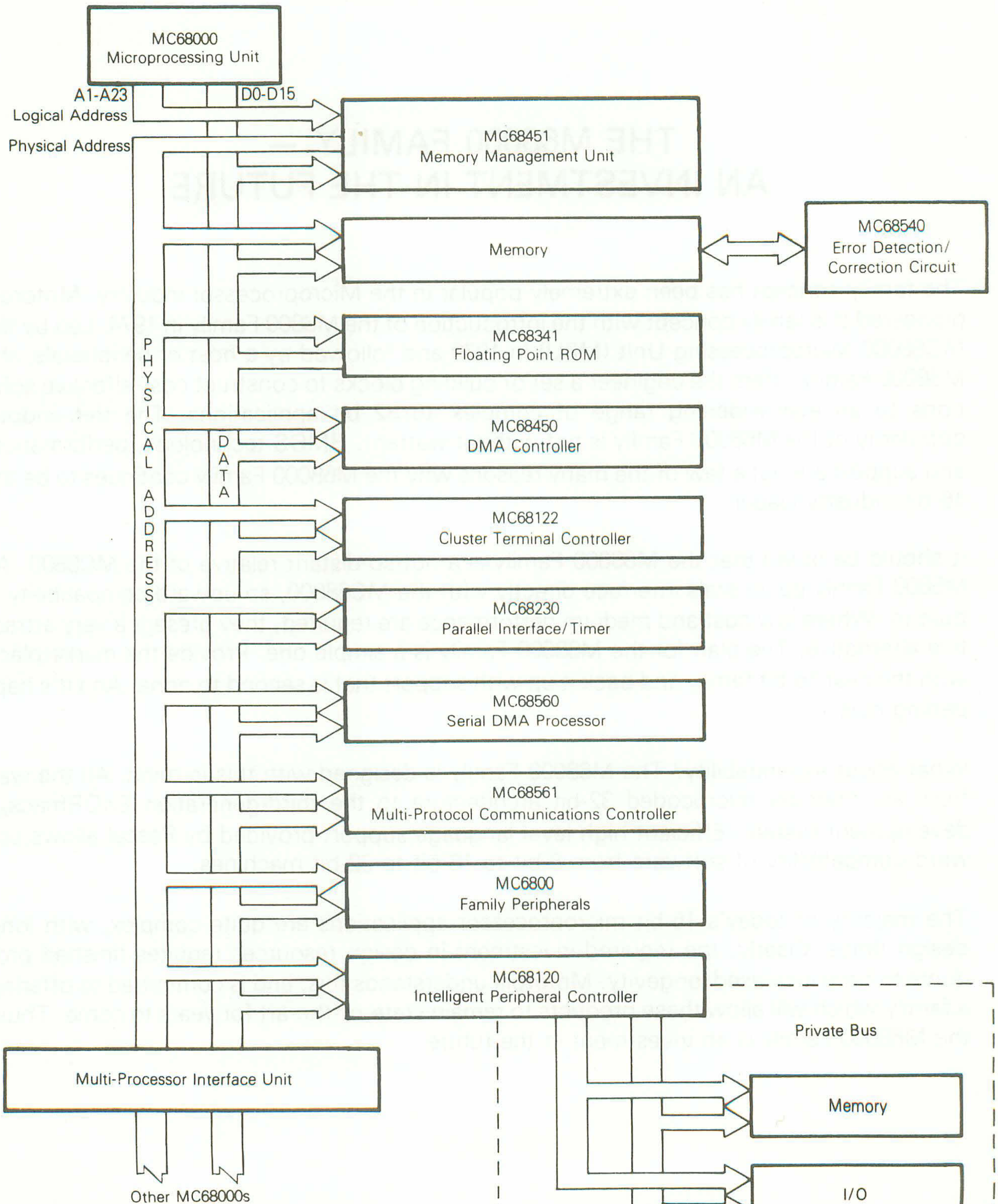
The family concept has been extremely popular in the Microprocessor industry. Motorola pioneered this family concept with the introduction of the M6800 Family in 1974. Led by the MC68000 Microprocessing Unit (MPU) in 1979 and followed by a host of peripherals, the M68000 Family offers the engineer a set of building blocks to construct cost-effective solutions to an ever-widening range of complex 16/32 bit applications. The tremendous popularity of the M68000 Family is not without warrant. HMOS technology, performance, and support are but a few of the many reasons why the M68000 Family continues to be the 16 bit industry leader.

It should be noted that the M68000 Family is a not-so-distant relative of the MC6800. All M6800 Family peripherals interface directly with the MC68000, so upward compatibility is built-in. Where low cost and medium performance are required, they present a very attractive alternative. The plan for the M68000 Family is a simple one. Provide the marketplace with the best 16-bit family and back it up with support that is second to none. And it's happening now.

What about expandability? The M68000 Family is designed with this in mind. All the way from an internally microcoded 32-bit architecture to the third-generation EXORmacs™ development system. Efficient high level language support provided by Pascal allows upward compatibility of software from 8-bit to 16-bit to 32-bit machines.

The majority of today's 16-bit microprocessor applications are quite complex, with long design times. Clearly, the required investment in design resources requires finished products to have increased longevity. Motorola understands this, and is committed to offering a family which will allow these products to remain state-of-the-art for years to come. Thus, the M68000 Family is an investment in the future.

EXPANDED VIEW OF THE LOCAL BUS



MOTOROLA

SEMICONDUCTORS

3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721

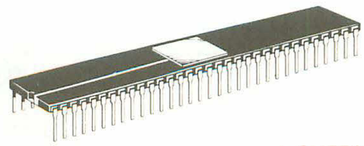
Advance Information

MC68000L4
(4 MHz)
MC68000L6
(6 MHz)
MC68000L
(8 MHz)

HMOS

(High Density N-Channel, Silicon-Gate Depletion Load)

16-BIT MICROPROCESSOR



L SUFFIX
CERAMIC PACKAGE
CASE 746-1

16-BIT MICROPROCESSING UNIT

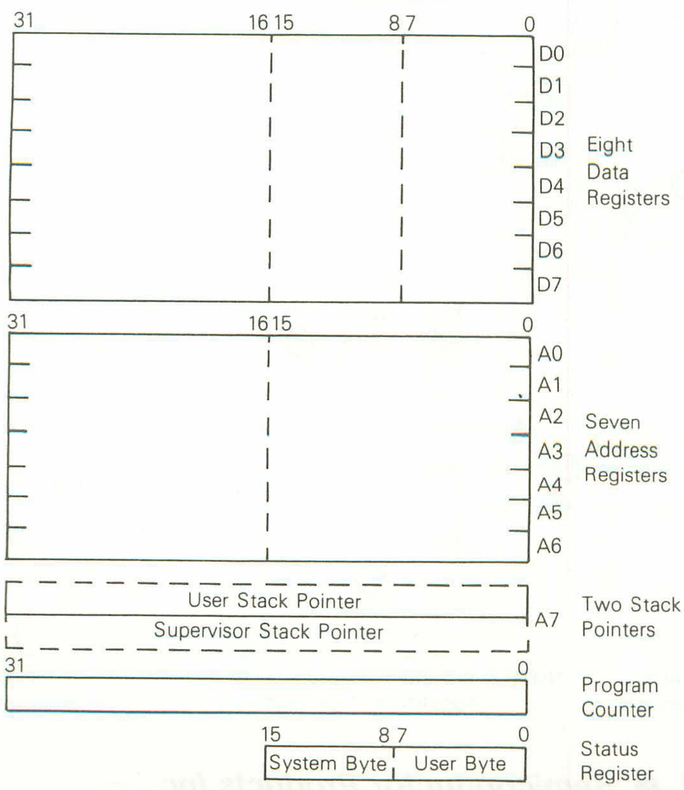
Advances in semiconductor technology have provided the capability to place on a single silicon chip a microprocessor at least an order of magnitude higher in performance and circuit complexity than has been previously available. The MC68000 is the first of a family of such VLSI microprocessors from Motorola. It combines state-of-the-art technology and advanced circuit design techniques with computer sciences to achieve an architecturally advanced 16-bit microprocessor.

The resources available to the MC68000 user consist of the following:

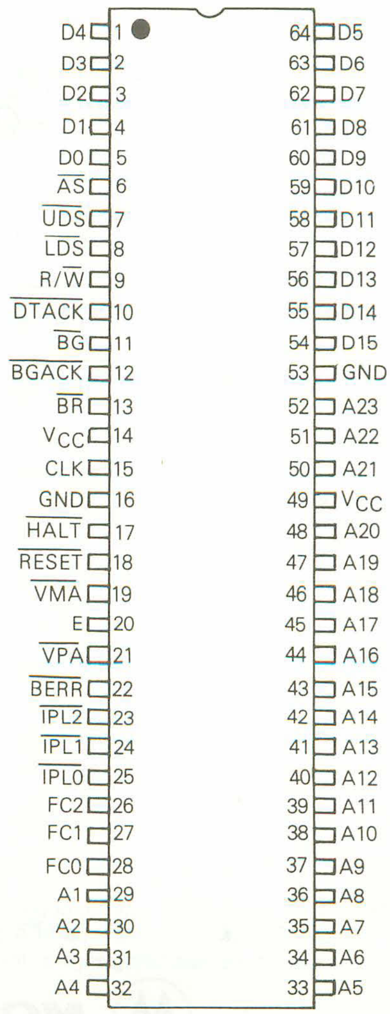
- 32-Bit Data and Address Registers
- 16 Megabyte Direct Addressing Range
- 56 Powerful Instruction Types
- Operations on Five Main Data Types
- Memory Mapped I/O
- 14 Addressing Modes

As shown in the programming model, the MC68000 offers seventeen 32-bit registers in addition to the 32-bit program counter and a 16-bit status register. The first eight registers (D0-D7) are used as data registers for byte (8-bit), word (16-bit), and long word (32-bit) data operations. The second set of seven registers (A0-A6) and the system stack pointer may be used as software stack pointers and base address registers. In addition, these registers may be used for word and long word address operations. All 17 registers may be used as index registers.

PROGRAMMING MODEL



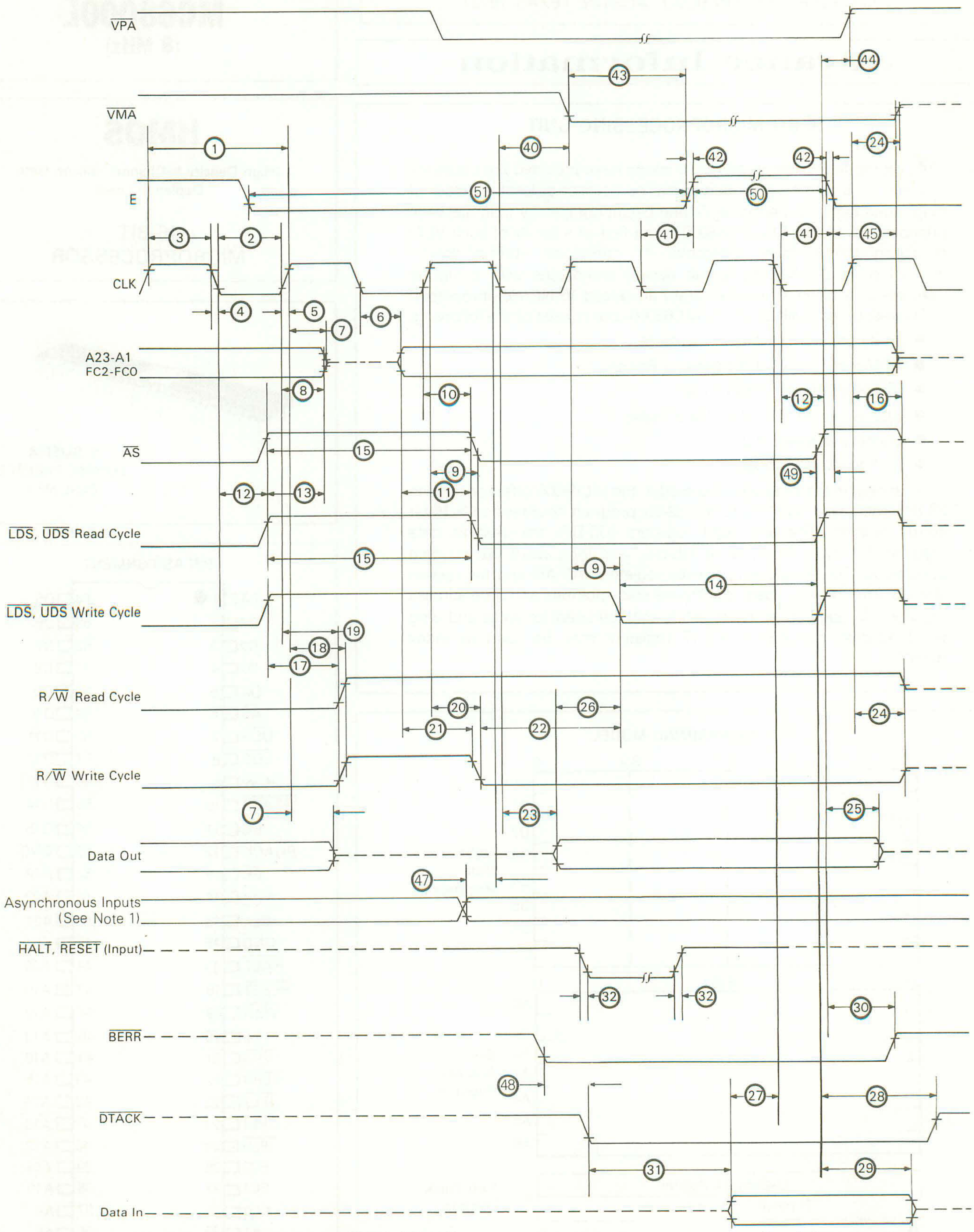
PIN ASSIGNMENT



This is advance information and specifications are subject to change without notice.

FIGURE 31 — AC ELECTRICAL WAVEFORMS

These waveforms should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional description of the input and output signals. Refer to other functional descriptions and their related diagrams for device operation.



NOTE 1: Setup time for the asynchronous inputs BERR, BGACK, BR, DTACK, IPL0-IPL2, and VPA guarantees their recognition at the next falling edge of the clock.

NOTE 2: Waveform measurements for all inputs and outputs are specified at: logic high = 2.0 volts, logic low = 0.8 volts.

AC ELECTRICAL SPECIFICATIONS (V_{CC} = 5.0 Vdc ± 5%; V_{SS} = 0 Vdc; T_A = 0°C to 70°C, Figure 31)

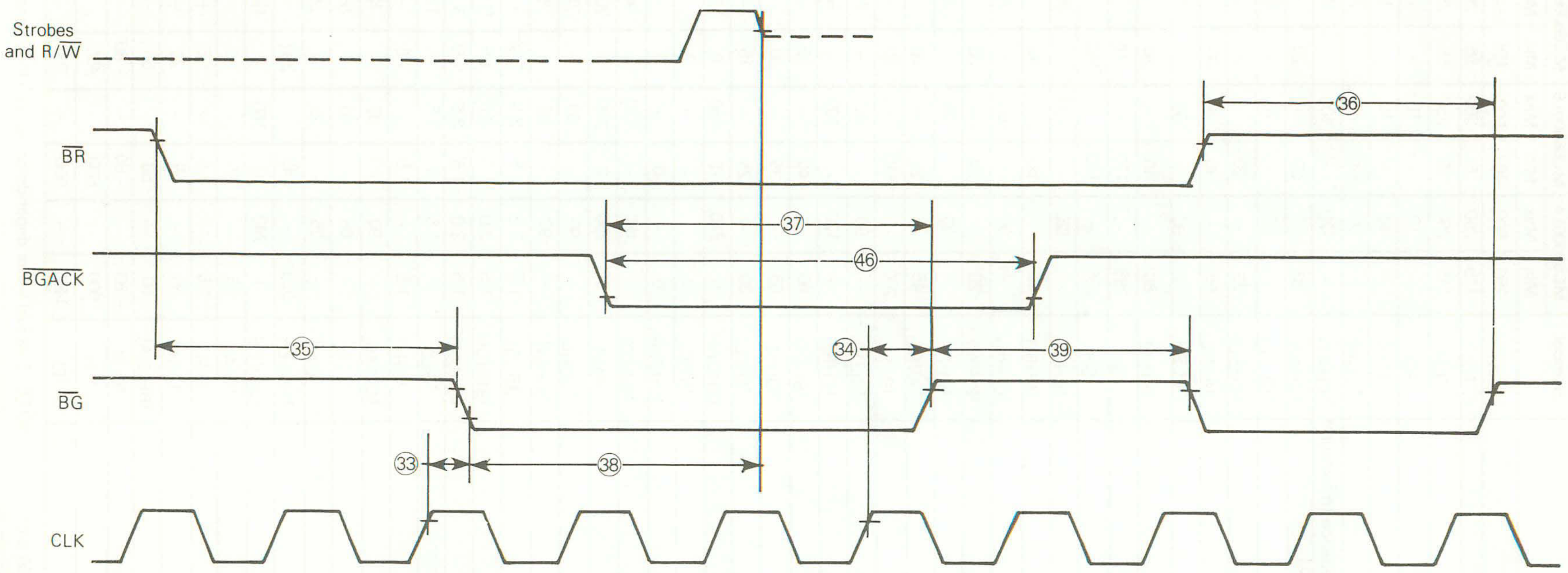
Number	Characteristic	Symbol	4 MHz MC68000L4		6 MHz MC68000L6		8 MHz MC68000L		Unit
			Min	Max	Min	Max	Min	Max	
1	Clock Period	t_{cyc}	250	500	167	500	125	500	ns
2	Clock Width Low	t_{CL}	115	250	75	250	55	250	ns
3	Clock Width High	t_{CH}	115	250	75	250	55	250	ns
4	Clock Fall Time	t_{Cf}	–	10	–	10	–	10	ns
5	Clock Rise Time	t_{Cr}	–	10	–	10	–	10	ns
6	Clock Low to Address/FC Valid	t_{CLAV}	–	90	–	80	–	70	ns
7	Clock High to Address/FC/Data High Impedance (maximum)	t_{CHAZx}	–	120	–	100	–	80	ns
8	Clock High to Address/FC Invalid (minimum)	t_{CHAZn}	20	–	20	–	20	–	ns
9[1]	Clock High to $\overline{AS}$, $\overline{DS}$ Low (maximum)	t_{CHSLx}	–	80	–	70	–	60	ns
10	Clock High to $\overline{AS}$, $\overline{DS}$ Low (minimum)	t_{CHSLn}	20	–	20	–	20	–	ns
11[2]	Address/FC Valid to $\overline{AS}$, $\overline{DS}$ (read) Low	t_{AVSL}	55	–	35	–	30	–	ns
12[1]	Clock Low to $\overline{AS}$, $\overline{DS}$ High	t_{CLSH}	–	90	–	80	–	70	ns
13[2]	$\overline{AS}$, $\overline{DS}$ High to Address/FC Invalid	t_{SHAZ}	60	–	40	–	30	–	ns
14[2]	$\overline{AS}$, $\overline{DS}$ Width Low	t_{SL}	285	–	170	–	115	–	ns
15[2]	$\overline{AS}$, $\overline{DS}$ Width High	t_{SH}	285	–	180	–	150	–	ns
16	Clock High to $\overline{AS}$, $\overline{DS}$ High Impedance	t_{CHSZ}	–	120	–	100	–	80	ns
17[2]	$\overline{DS}$ High to R/$\overline{W}$ High	t_{SHRH}	60	–	50	–	40	–	ns
18[1]	Clock High to R/$\overline{W}$ High (maximum)	t_{CHRHx}	–	90	–	80	–	70	ns
19	Clock High to R/$\overline{W}$ High (minimum)	t_{CHRHn}	20	–	20	–	20	–	ns
20[1]	Clock High to R/$\overline{W}$ Low	t_{CHRL}	–	90	–	80	–	70	ns
21[2]	Address/FC Valid to R/$\overline{W}$ Low	t_{AVRL}	45	–	25	–	20	–	ns
22[2]	R/$\overline{W}$ Low to $\overline{DS}$ Low (write)	t_{RLSL}	200	–	140	–	80	–	ns
23	Clock Low to Data Out Valid	t_{CLDO}	–	90	–	80	–	70	ns
24	Clock High to R/$\overline{W}$, $\overline{VMA}$ High Impedance	t_{CHRZ}	–	120	–	100	–	80	ns
25[2]	$\overline{DS}$ High to Data Out Invalid	t_{SHDO}	60	–	40	–	30	–	ns
26[2]	Data Out Valid to $\overline{DS}$ Low (write)	t_{DOSL}	55	–	35	–	30	–	ns
27	Data In to Clock Low (set up time)	t_{DICL}	30	–	25	–	15	–	ns
28[2]	$\overline{DS}$ High to $\overline{DTACK}$ High	t_{SHDAH}	0	240	0	160	0	120	ns
29	$\overline{DS}$ High to Data Invalid (hold time)	t_{SHDI}	0	–	0	–	0	–	ns
30	$\overline{AS}$, $\overline{DS}$ High to $\overline{BERR}$ High	t_{SHBEH}	0	–	0	–	0	–	ns
31[2]	$\overline{DTACK}$ Low to Data In (setup time)	t_{DALDI}	–	180	–	120	–	90	ns
32	$\overline{HALT}$ and $\overline{RESET}$ Input Transition Time	t_{RHrf}	0	200	0	200	0	200	ns
33	Clock High to $\overline{BG}$ Low	t_{CHGL}	–	90	–	80	–	70	ns
34	Clock High to $\overline{BG}$ High	t_{CHGH}	–	90	–	80	–	70	ns
35	$\overline{BR}$ Low to $\overline{BG}$ Low	t_{BRLGL}	1.5	3.0	1.5	3.0	1.5	3.0	clk. per.
36	$\overline{BR}$ High to $\overline{BG}$ High	t_{BRHGH}	1.5	3.0	1.5	3.0	1.5	3.0	clk. per.
37	$\overline{BGACK}$ Low to $\overline{BG}$ High	t_{GALGH}	1.5	3.0	1.5	3.0	1.5	3.0	clk. per.
38	$\overline{BG}$ Low to Bus High Impedance (with $\overline{AS}$ high)	t_{GLZ}	0	1.5	0	1.5	0	1.5	clk. per.
39	$\overline{BG}$ Width High	t_{GH}	1.5	–	1.5	–	1.5	–	clk. per.
40	Clock Low to $\overline{VMA}$ Low	t_{CLVML}	–	90	–	80	–	70	ns
41	Clock Low to E Transition	t_{CLE}	–	65	–	60	–	55	ns
42	E Output Rise and Fall Time	t_{Erf}	–	25	–	25	–	25	ns
43[2]	$\overline{VMA}$ Low to E High	t_{VMLEH}	325	–	240	–	200	–	ns
44	$\overline{AS}$, $\overline{DS}$ High to $\overline{VPA}$ High	t_{SHVPH}	0	240	0	160	0	120	ns
45	E Low to Address/$\overline{VMA}$/FC Invalid	t_{ELAI}	55	–	35	–	30	–	ns
46	$\overline{BGACK}$ Width	t_{BGL}	1.5	–	1.5	–	1.5	–	clk. per.
47	Asynchronous Input Setup Time	t_{ASI}	30	–	25	–	20	–	ns
48	$\overline{BERR}$ Low to $\overline{DTACK}$ Low	t_{BELDAL}	50	–	50	–	50	–	ns
49	E Low to $\overline{AS}$, $\overline{DS}$ Invalid	t_{ELSI}	−80	–	−80	–	−80	–	ns
50	E Width High	t_{EH}	900	–	600	–	450	–	ns
51	E Width Low	t_{EL}	1400	–	900	–	700	–	ns

NOTE 1: For a loading capacitance of less than or equal to 50 picofarads, subtract 5 nanoseconds from the values given in these columns.

NOTE: 2 Actual value depends on actual clock period.

FIGURE 32 — AC ELECTRICAL WAVEFORMS — BUS ARBITRATION

These waveforms should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional description of the input and output signals. Refer to other functional descriptions and their related diagrams for device operation.



AC ELECTRICAL SPECIFICATIONS — BUS ARBITRATION (V_{DD} = 5.0 Vdc ± 5%; V_{SS} = 0 Vdc; T_A = 0°C to 70°C, Figure 32)

Number	Characteristic	Symbol	4 MHz MC68000L4		6 MHz MC68000L6		8 MHz MC68000L		Unit
			Min	Max	Min	Max	Min	Max	
33	Clock High to $\overline{BG}$ Low	t_{CHGL}	–	90	–	80	–	70	ns
34	Clock High to $\overline{BG}$ High	t_{CHGH}	–	90	–	80	–	70	ns
35	$\overline{BR}$ Low to $\overline{BG}$ Low	t_{BRLGL}	1.5	3.0	1.5	3.0	1.5	3.0	clk. per.
36	$\overline{BR}$ High to $\overline{BG}$ High	t_{BRHGH}	1.5	3.0	1.5	3.0	1.5	3.0	clk. per.
37	$\overline{BGACK}$ Low to $\overline{BG}$ High	t_{GALGH}	1.5	3.0	1.5	3.0	1.5	3.0	clk. per.
38	$\overline{BG}$ Low to Bus High Impedance (with $\overline{AS}$ high)	t_{GLZ}	0	1.5	0	1.5	0	1.5	clk. per.
39	$\overline{BG}$ Width High	t_{GH}	1.5	–	1.5	–	1.5	–	clk. per.
46	$\overline{BGACK}$ Width	t_{BGL}	1.5	—	1.5	—	1.5	—	clk. per.

NOTE 1: Setup time for the asynchronous inputs $\overline{BERR}$, $\overline{BGACK}$, $\overline{BR}$, $\overline{DTACK}$, $\overline{IPL0}$-$\overline{IPL2}$, and $\overline{VPA}$ guarantees their recognition at the next falling edge of the clock.

NOTE 2: Waveform measurements for all inputs and outputs are specified at: logic high = 2.0 volts, logic low = 0.8 volts.

ELECTRICAL CHARACTERISTICS (V_{CC} = 5.0 Vdc ± 5%; V_{SS} = 0 Vdc; T_A 0°C to 70°C, Figures 33, 34, 35)

Characteristic		Symbol	Min	Typ	Max	Unit
Input High Voltage		V_{IH}	2.0	–	V_{CC}	Vdc
Input Low Voltage		V_{IL}	V_{SS} – 0.3	–	0.8	Vdc
Input Leakage Current	$\overline{\text{BERR}}$, $\overline{\text{BGACK}}$, $\overline{\text{BR}}$, $\overline{\text{DTACK}}$, $\overline{\text{IPL0}}$-$\overline{\text{IPL2}}$, $\overline{\text{VPA}}$	I_{in}	–	1.0	–	μAdc
	$\overline{\text{HALT}}$, $\overline{\text{RESET}}$		–	2.0	–	
Three-State (Off State) Input Current	$\overline{\text{AS}}$, A1-A23, D0-D15, FC0-FC2, $\overline{\text{LDS}}$, R/$\overline{\text{W}}$, $\overline{\text{UDS}}$, $\overline{\text{VMA}}$	I_{TSI}	–	7.0	–	μAdc
Output High Voltage (I_{OH} = –400 μAdc)	$\overline{\text{AS}}$, A1-A23, $\overline{\text{BG}}$, D0-D15, E, FC0-FC2, $\overline{\text{LDS}}$, R/$\overline{\text{W}}$, $\overline{\text{UDS}}$, $\overline{\text{VMA}}$	V_{OH}	2.4	–	–	Vdc
Output Low Voltage (I_{OL} = 1.6 mA)	$\overline{\text{HALT}}$	V_{OL}	–	–	0.5	Vdc
(I_{OL} = 3.2 mA)	A1-A23, $\overline{\text{BG}}$, E, FC0-FC2		–	–	0.5	
(I_{OL} = 5.0 mA)	$\overline{\text{RESET}}$		–	–	0.5	
(I_{OL} = 5.3 mA)	$\overline{\text{AS}}$, D0-D15, $\overline{\text{LDS}}$, R/$\overline{\text{W}}$, $\overline{\text{UDS}}$, $\overline{\text{VMA}}$		–	–	0.5	
Power Dissipation (Clock Frequency = 8 MHz)		P_D	–	1.0	–	W
Capacitance (Package Type Dependent) (V_{in} = 0 Vdc; T_A = 25°C; Frequency = 1 MHz)		C_{in}	–	10.0	–	pF

FIGURE 33 — RESET TEST LOAD

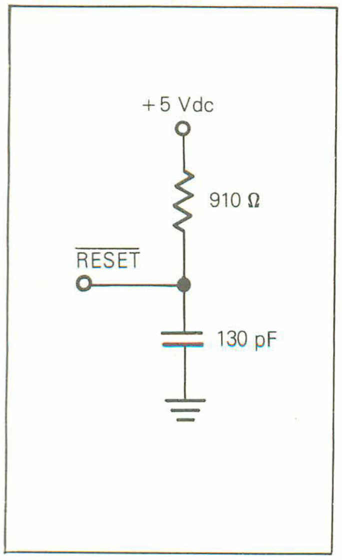


FIGURE 34 — HALT TEST LOAD

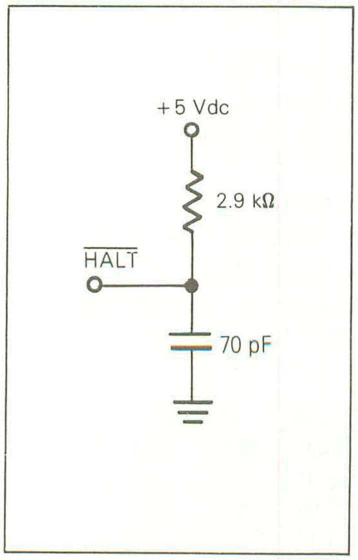


FIGURE 35 — TEST LOADS

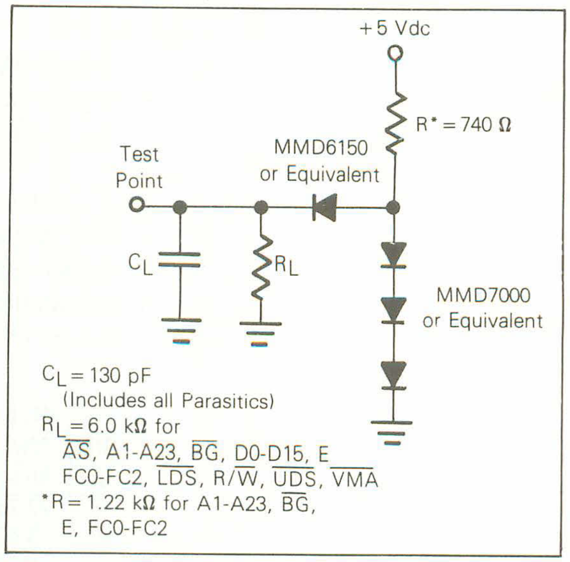
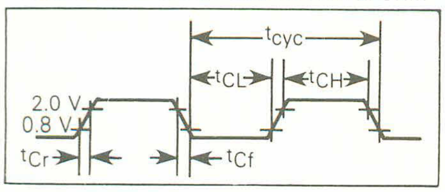


FIGURE 36 — INPUT CLOCK WAVEFORM

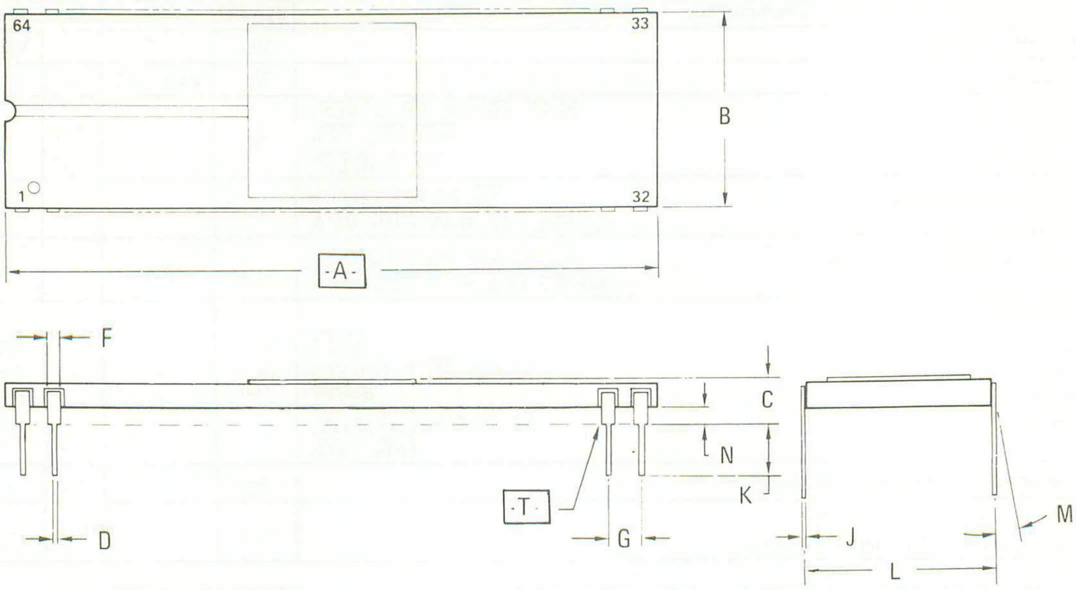


MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V_{CC}	–0.3 to +7.0	Vdc
Input Voltage	V_{in}	–0.3 to +7.0	Vdc
Operating Temperature Range	T_A	0 to 70	°C
Storage Temperature	T_{stg}	–55 to 150	°C

CLOCK TIMING (Figure 36)

Characteristic	Symbol	4 MHz MC6800L4		6 MHz MC68000L6		8 MHz MC68000L		Unit
		Min	Max	Min	Max	Min	Max	Unit
Frequency of Operation	F	2.0	4.0	2.0	6.0	2.0	8.0	MHz
Cycle Time	t_{cyc}	250	500	167	500	125	500	ns
Clock Pulse Width	t_{CL}	115	250	75	250	55	250	ns
	t_{CH}	115	250	75	250	55	250	
Rise and Fall Times	t_{Cr}	–	10	–	10	–	10	ns
	t_{Cf}	–	10	–	10	–	10	



NOTES:
1. DIMENSION -A- IS DATUM.
2. POSITIONAL TOLERANCE FOR LEADS:
   ⊕ 0.25 (0.010) Ⓜ T A Ⓜ
3. -T- IS SEATING PLANE.
4. DIMENSION "L" TO CENTER OF LEADS WHEN FORMED PARALLEL.
5. DIMENSIONING AND TOLERANCING PER ANSI Y14.5, 1973.

DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	80.52	82.04	3.170	3.230
B	22.25	22.96	0.876	0.904
C	3.05	4.32	0.120	0.170
D	0.38	0.53	0.015	0.021
F	0.76	1.40	0.030	0.055
G	2.54 BSC		0.100 BSC	
J	0.20	0.33	0.008	0.013
K	2.54	4.19	0.100	0.165
L	22.61	23.11	0.890	0.910
M	–	10°	–	10°
N	1.02	1.52	0.040	0.060

CASE 746-01

MOTOROLA
SEMICONDUCTORS
3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721

MC68120

Product Description

INTELLIGENT PERIPHERAL CONTROLLER

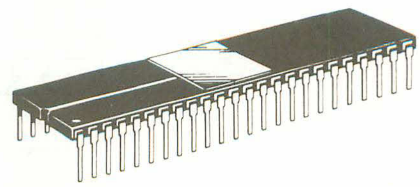
The MC68120 Intelligent Peripheral Controller (IPC) is a general-purpose, user-programmable, peripheral controller. It contains a system interface, an 8-bit CPU, a serial communications interface, 21 parallel I/O lines, a 16-bit timer, 2048 bytes of ROM, and eight operating modes. In addition, the MC68120 features 128 bytes of dual-ported RAM and six semaphore registers that are accessable to both the internal CPU and an external processor or device through the system interface. The MC68120 provides all the control signals necessary to interface with the asynchronous bus of the MC68000.

- Bus Compatible with the 16-Bit MC68000 and the 8-Bit MC6809 Microprocessors
- Bus Compatible with all M6800 Processors and Peripherals
- TTL Compatible I/O
- Single 5-Volt Power Supply
- 8-Bit CPU
- 2K Bytes ROM
- 128 Bytes Dual-Ported RAM
- 6 Shared Semaphore Registers
- External Clock Input
- 16-Bit Timer
- 21 Parallel I/O and Two Handshake Lines
- Serial Communications Interface
- Interrupt Capability
- Operates in Single-Chip Mode or Expandable to 64K Byte Addressing Range
- DMA Capability
- Upward Software Compatibility with MC6800, MC6801, MC6802, and MC6803 Processors
- Upward Source and Object Code Compatibility with MC6800 and MC6801 Processors
- 8 X 8 Bit Multiply
- Software Control of:
  - Semaphore Registers
  - 16-Bit Timer
  - Serial Communications Interface
  - Parallel I/O Ports
  - Interrupts

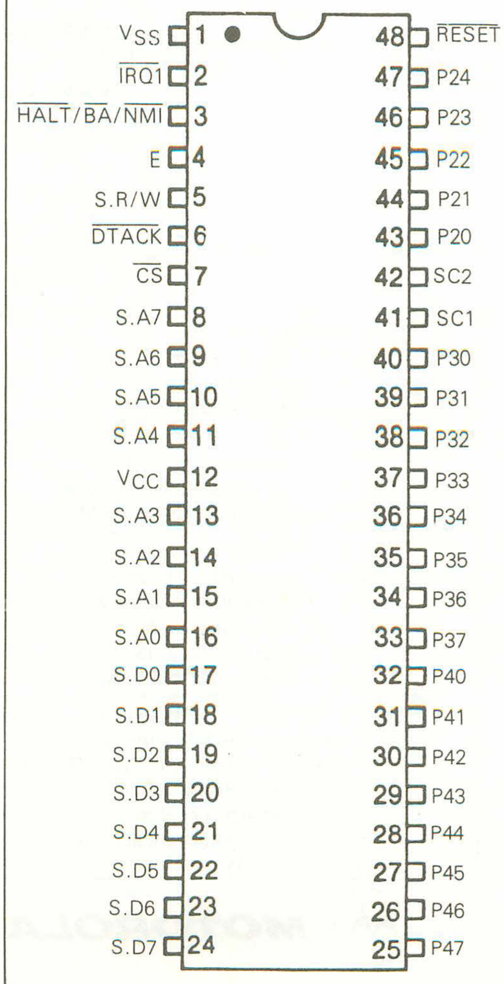
HMOS

(HIGH DENSITY N-CHANNEL, SILICON-GATE DEPLETION LOAD)

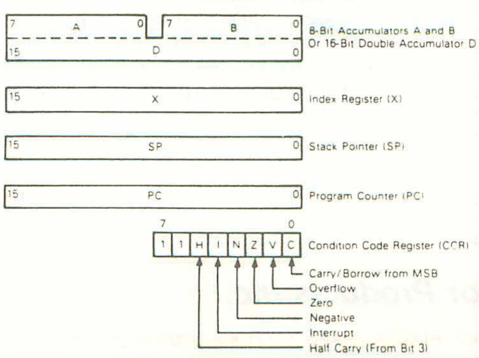
INTELLIGENT PERIPHERAL CONTROLLER



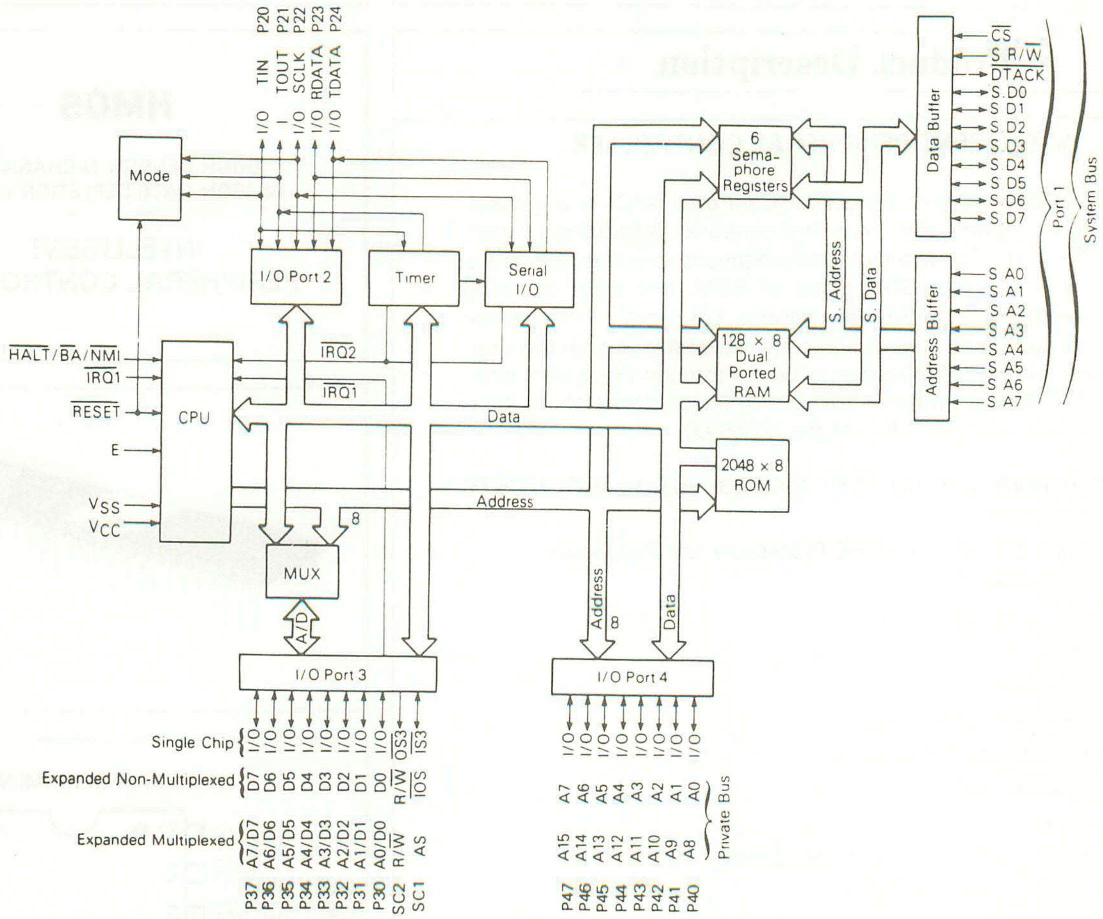
PIN ASSIGNMENT



PROGRAMMING MODEL



BLOCK DIAGRAM



SUMMARY OF OPERATING MODES

Common to all Modes:
- System Bus Interface
- Reserved Register Area
- 6 Semaphore Registers
- I/O Port 2
- 16-Bit Programmable Timer
- Serial Communications Interface
- 128 Bytes of Dual-Ported RAM

Single Chip Mode — Mode 7
- 2048 Bytes of ROM (Internal)
- Port 3 is a Parallel I/O Port with Two Control Lines
- Port 4 is a Parallel I/O Port
- SC1 is Input Strobe 3 (IS3)
- SC2 is Output Strobe 3 (OS3)

Expanded Non-Multiplexed Mode — Mode 5
- 2048 Bytes of ROM (Internal)
- 256 Bytes of External Memory Space
- Port 3 is an 8-Bit Data Bus
- Port 4 is an Address Bus
- SC1 is Input/Output Select (IOS)
- SC2 is Read/Write (R/W)

Expanded Multiplexed Modes — Modes 1, 2, 3, 6
- Four Memory Space Options (64K Address Space):
  - (1) MDOS Compatible
  - (2) No ROM
  - (3) External Vector Space
  - (4) ROM with Partial Address Bus
- External Memory Space Accessed Through:
  - Port 3 as a Multiplexed Address/Data Bus
  - Port 4 as an Address Bus (High)
- SC1 is Address Strobe (AS) Input
- SC2 is Read/Write (R/W)

Test Modes — Modes 0, 4
- Expanded Multiplexed Test Mode —
  - May be Used to Test RAM and ROM
- Single Chip and Non-Multiplexed Test Mode —
  - May be Used to Test Ports 3 and 4 as I/O Ports

MOTOROLA

SEMICONDUCTORS

3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721

MC68122

Product Description

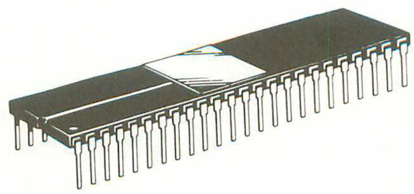
CLUSTER TERMINAL CONTROLLER

The MC68122 Cluster Terminal Controller (CTC) relieves the host processor of the task of coordinating the communications between one or more terminals. Messages containing either text or control commands are passed between the host processor and the controller according to the requirements of the application program being run. Standard terminal devices and protocols are supported. Output- or input-only devices such as line printers may also be attached. The CTC supports its own local bus for message buffering and interface to communications devices, or operating in a single device mode, supports a single serial port.

HMOS

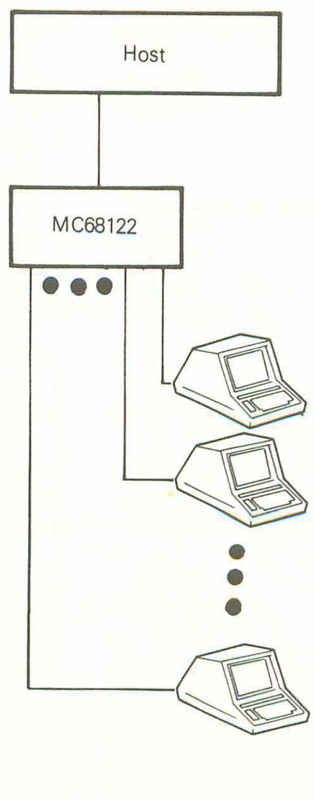
(HIGH DENSITY N-CHANNEL, SILICON-GATE DEPLETION LOAD)

CLUSTER TERMINAL CONTROLLER



- Compatible with M68000 and M6800 Families
- Allows Complete Control by Host Processor
- Broadcast Feature Provides Automatic Generation of Text to all Terminals
- Host Processor may Examine and/or Change any Portion of the Controller's Local Memory
- Complete Status Reporting Capability
- Echo Control
- Carriage Return Options
- Supports Automatic Error Recovery
- Single 5-Volt Power Supply

TYPICAL TERMINAL INTERCONNECTIONS



MOTOROLA
SEMICONDUCTORS
BOX 20912 • PHOENIX, ARIZONA 85036

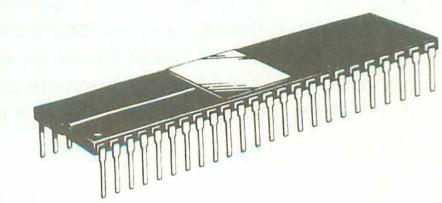
MC68540

Product Description

ERROR DETECTION AND CORRECTION CIRCUIT

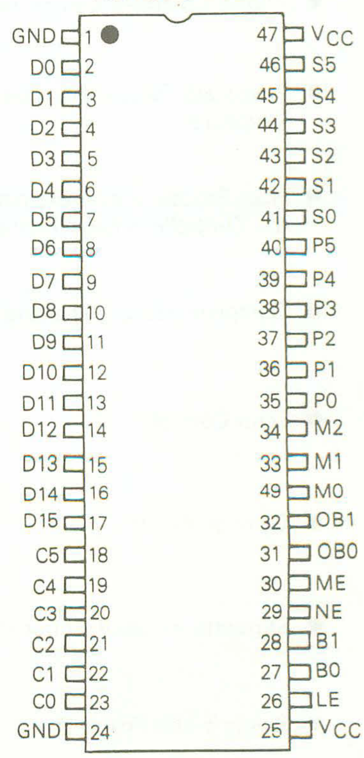
(SILICON MONOLITHIC INTEGRATED CIRCUIT)

The MC68540 Error Detection and Correction Circuit (EDAC) performs the error check/correct function in both 8-bit and 16-bit data bus systems and is direclty expandable to 32-bit operation. The EDAC contains an internal diagnostic register and a set of flexible diagnostic operating modes to facilitate system diagnostic testing and error logging. All required bus buffering is handled within the device to reduce the need for support circuitry.

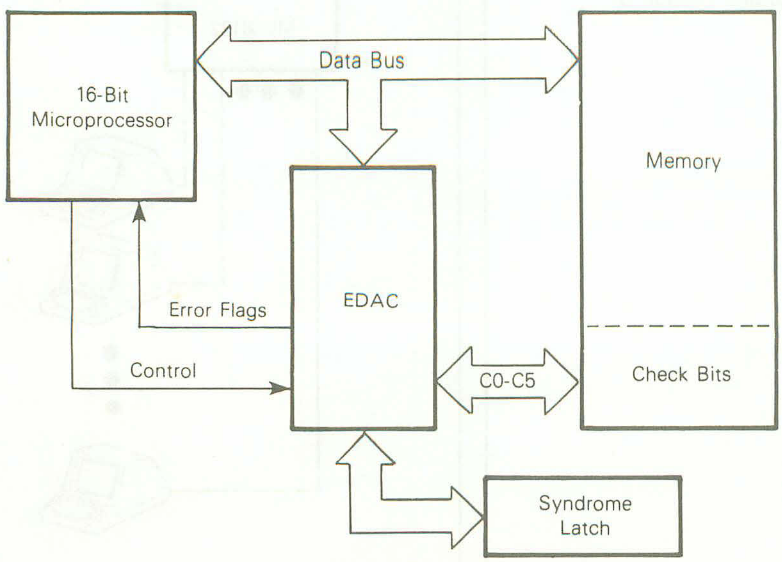


- Single Device Provides 8-Bit and 16-Bit Operation
- Two Devices Provide 32-Bit Operation
- Bidirectional Data and Check Bits
- Single Error Correction; Double Error Detection
- Detects Catastrophic Error of All Ones or All Zeros
- Independent Byte Control
- Supports Parity (Read and Write)
- Error Flags
- Flexible Diagnostic Operating Modes
- Internal Diagnostic Register
- Syndromes Available for Hardware Latching
- Input and Output Data Latched

PIN ASSIGNMENT



TYPICAL CONFIGURATION



MOTOROLA
SEMICONDUCTORS
3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721

MC68451

Product Description

MEMORY MANAGEMENT UNIT

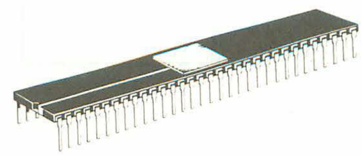
The MC68451 Memory Management Unit (MMU) provides address translation and protection of the 16 megabyte addressing space of the MC68000. The MMU can be accessed by any potential bus master, such as instruction set processors, or DMA controllers. Each bus master (or processor) in the M68000 family provides a function code and an address during each bus cycle. The function code specifies an address space while the address specifies a location within that address space. The function codes are provided by the MC68000 to distinguish between program and data spaces as well as supervisor and user spaces. This separation of address spaces provides the basis of protection in an operating system. By simplifying the programming model of the address space, the MMU also increases the reliability of a complex multi-process system.

- Separates Address Spaces of System and User Resources
- Provides Write Protection
- Increases System Reliability
- Provides Efficient Memory Allocation
- Allows Interprocess Communication through Shared Resources
- Simplifies Programming Model of Address Space
- Minimizes Operating System Overhead with Quick Context Switches
- 32 Segments with Variable Segment Sizes
- Multiple MMU System Capability
- Supports both Paging and Segmentation
- DMA Compatible
- Provides Virtual Memory Support
- MC68000 Bus Compatible

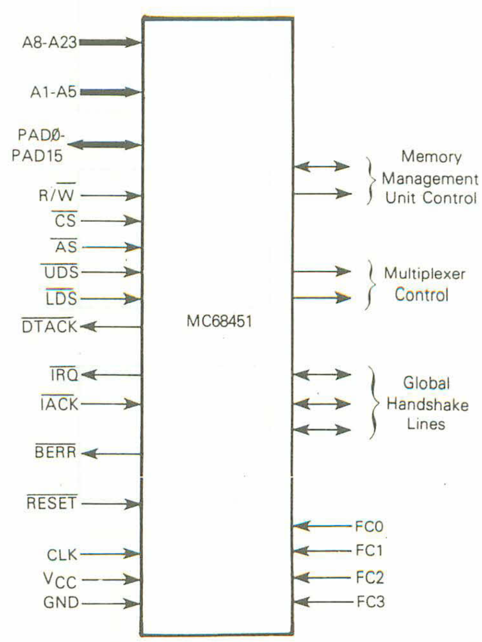
HMOS

(HIGH DENSITY N-CHANNEL, SILICON-GATE DEPLETION LOAD)

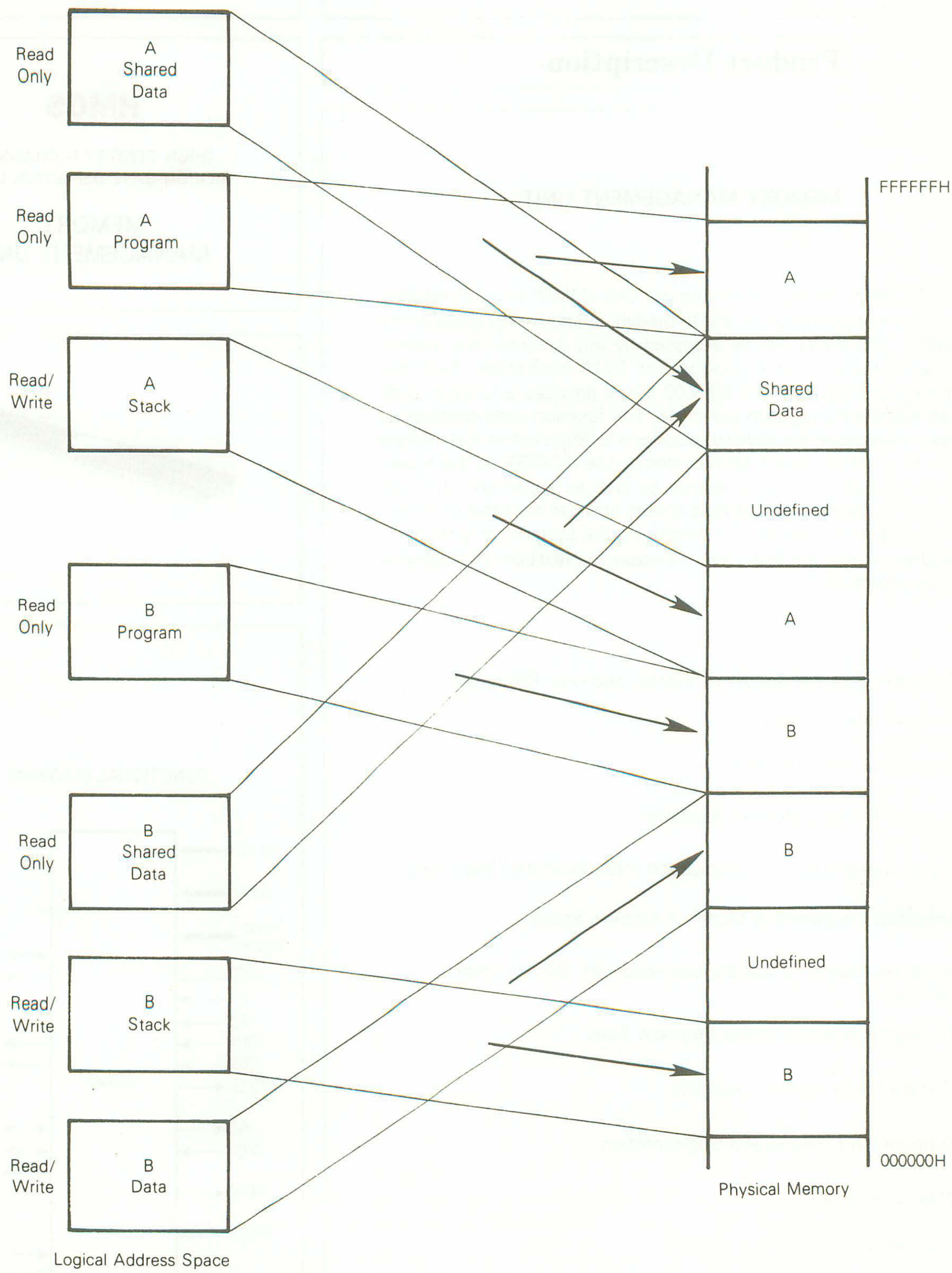
MEMORY MANAGEMENT UNIT



FUNCTIONAL DIAGRAM



MAPPING LOGICAL SEGMENTS TO PHYSICAL MEMORY



MOTOROLA
SEMICONDUCTORS
3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721

MC68450

Product Description

DIRECT MEMORY ACCESS CONTROLLER

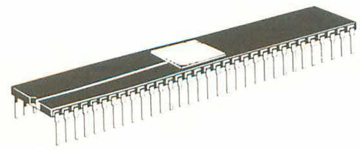
The MC68450 Direct Memory Access Controller (DMAC) offers the system designer unparalleled performance where both speed and flexibility in data transfer are required. Sophisticated chaining techniques, memory-to-memory block transfers, and variable bus bandwith utilization result in optimum data transfers. Internal 32-bit address registers provide upward software compatibility with future M68000 Family processors.

- Compatible with both M68000 Family and M6800 Peripherals
- Four, Fully-Independent Channels
- Single or Dual Address Transfers
- Byte, Word, or Long Word Transfers
- Memory-to-Memory Block Transfers
- Supports both Chained and Unchained Operations
- Transfer Rates up to 4 Megabytes Per Second
- Supports Vectored Interrupts
- Supports Array or Linked Array Chaining
- 16-Bit Data Bus
- Programmable Priorities

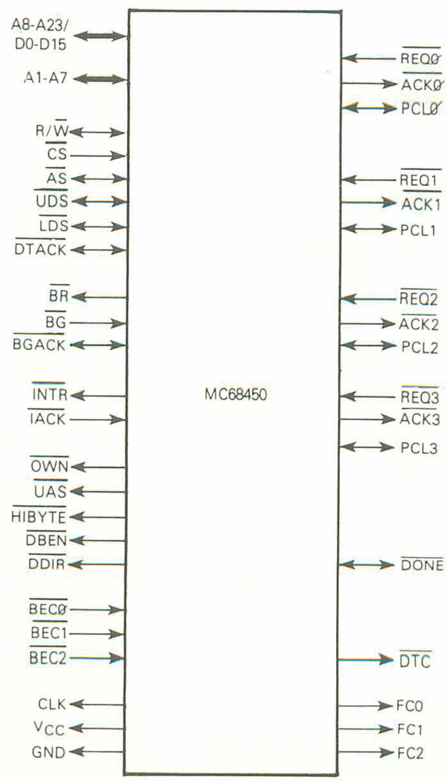
HMOS

(HIGH DENSITY N-CHANNEL, SILICON-GATE DEPLETION LOAD)

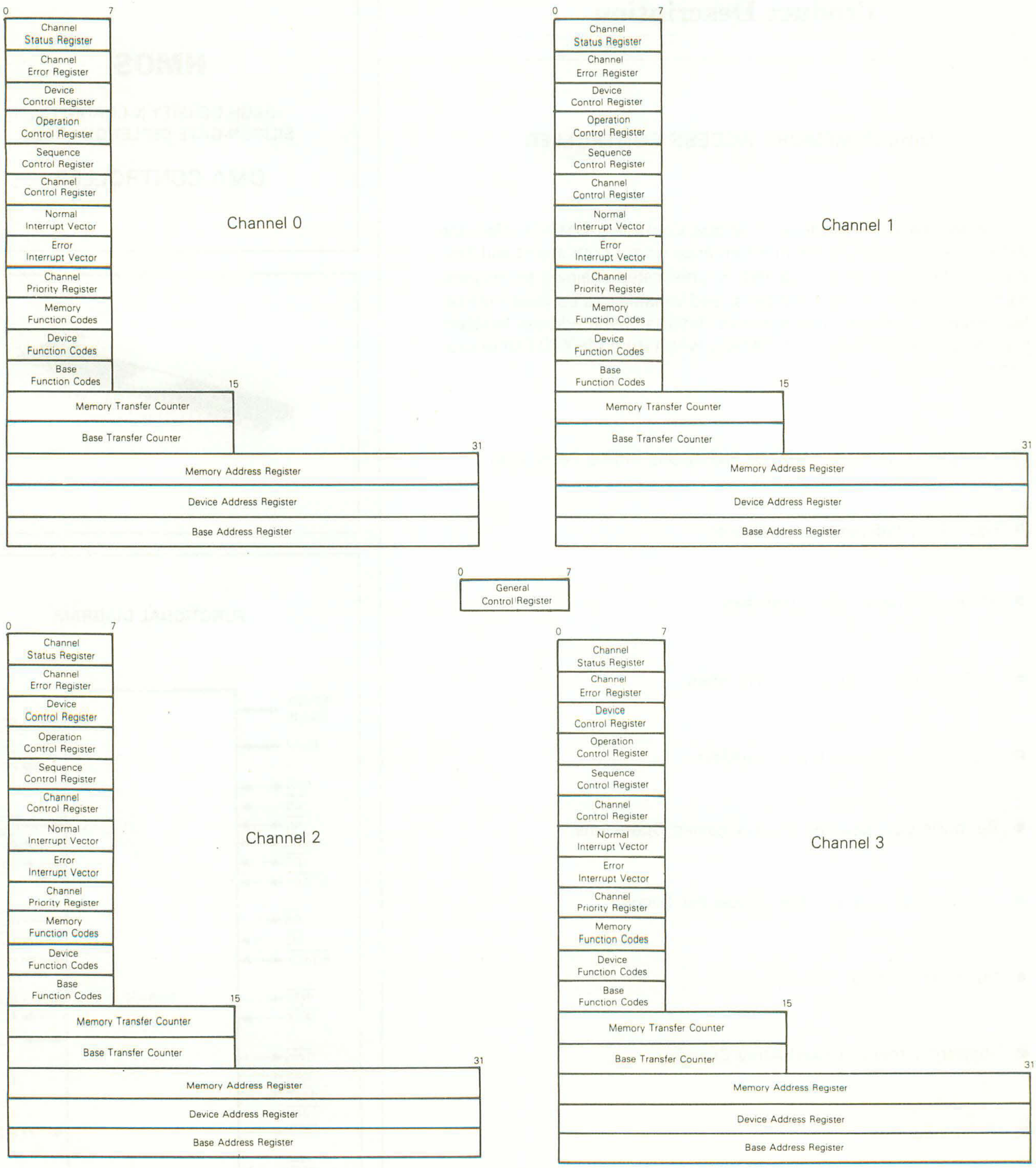
DMA CONTROLLER



FUNCTIONAL DIAGRAM



INTERNAL ORGANIZATION



MOTOROLA
SEMICONDUCTORS
3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721

MC68230

Product Description

HMOS

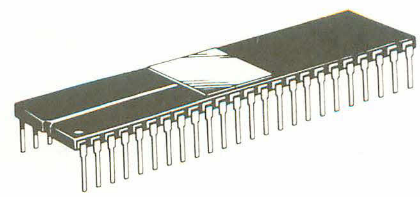
(HIGH DENSITY N-CHANNEL, SILICON-GATE DEPLETION LOAD)

PARALLEL INTERFACE/TIMER

PARALLEL INTERFACE/TIMER

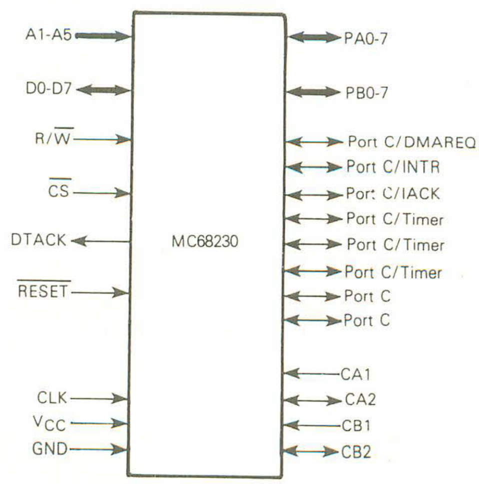
The MC68230 Parallel Interface/Timer (PI/T) is a general purpose programmable parallel interface device which meets most system parallel I/O and timing needs. The parallel interfaces of the PI/T may be used to connect a wide variety of peripheral devices to the MC68000 bus. The PI/T can be programmed to generate interrupt requests to the MC68000, or DMA requests to the MC68450 Direct Memory Access Controller, on demand of the peripheral. No external logic is needed to connect the PI/T to the DMAC or to the MC68000. The PI/T also provides complete port and timer status information.

The MC68230 contains two multi-mode double-buffered I/O ports (Port A and Port B), a third 8-bit I/O port, and a 24-bit programmable timer. The PI/T also contains flexible prioritization logic for generating unique interrupt vectors. Autovectored interrupts are also supported. Port C pins provide either another port pin or special functions consisting of interrupt, timer, or Direct Memory Access control lines.



- 24 Programmable I/O Lines
- Port Modes Include:
  - Bit Mode
  - Unidirectional 8-Bit Mode
  - Unidirectional 16-Bit Mode
  - Bidirectional 8-Bit Mode
  - Bidirectional 16-Bit Mode
- Selectable Handshaking Modes
- 24-Bit Programmable Timer
- Several Timer Modes
- Contains Interrupt Prioritization Logic
- Unique Interrupt Vectors for Each Source
- Supports Interrupt and DMA Service Requests
- Contains Port and Timer Status Registers
- Registers are Read/Write and Directly Addressable
- Registers Addressed for MOVEP and DMAC Compatibility
- MC68000 Bus Compatible

FUNCTIONAL DIAGRAM



MOTOROLA

SEMICONDUCTORS

3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721

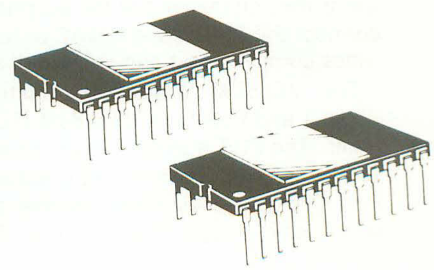
MC68341

Product Description

HMOS

(HIGH DENSITY N-CHANNEL, SILICON-GATE DEPLETION LOAD)

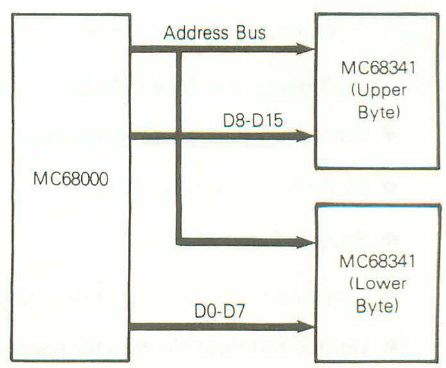
FLOATING-POINT ROM



FLOATING-POINT ROM

- Position Independent/Re-Entrant Floating-Point ROM
- Programmed for the MC68000 Instruction Set
- Uses the Unimplemented Instruction Trap to Simulate Hardware Floating Point
- Compatible with the Proposed IEEE Standard
- Single, Double, and Extended Formats
- Includes the Following Operations:
  - Add
  - Subtract
  - Multiply
  - Divide
  - Remainder
  - Square Root
  - Compare
  - Absolute Value
  - Negate
  - Integer Part
  - Convert Integer ↔ Floating Point
  - Convert Binary ↔ Decimal
- All Operations Done to Extended Precision
- 2 Device Set (Upper Byte and Lower Byte)

TYPICAL CONFIGURATION



MOTOROLA

SEMICONDUCTORS

3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721

MC68561

Product Description

MULTI-PROTOCOL COMMUNICATIONS CONTROLLER

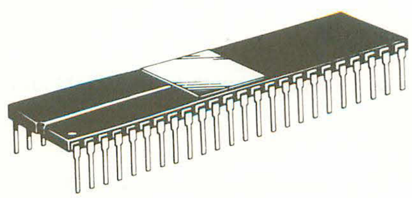
The MC68561 Multi-Protocol Communication Controller (MPCC) is a serial data communications interface for the M68000 Family. This device meets the basic interface requirements of asynchronous, bit- and byte-oriented synchronous communication protocols. To keep device count low, the MPCC also contains an internal crystal oscillator and baud rate generator.

- Complete Data Communications Interface
- Line Protocols:
  Asynchronous
  Bit-Oriented Synchronous (X.25, SDLC, HDLC)
  Byte-Oriented Synchronous (BISYNC, DDCMP)
- Direct Interface to MC68000
- MPU Interface Compatible with MC68450 DMAC
- Self Test Loop Mode
- Single 5-Volt Power Supply
- Full or Half Duplex Operation
- Multiple CRC Character Generation and Error Detection
- Internal Baud Rate Generator
- Internal Crystal Oscillator
- Complete Status Reporting Capability
- Buffered Transmit and Receive Registers
- Supports Vectored Interrupts
- MC68000 Bus Compatible

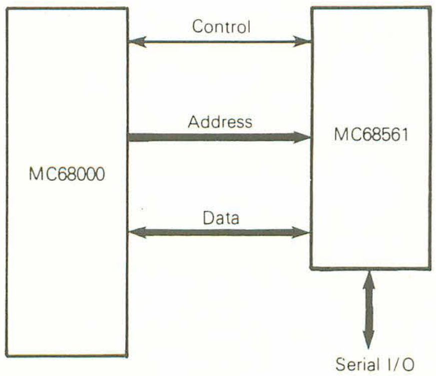
HMOS

(HIGH DENSITY N-CHANNEL, SILICON-GATE DEPLETION LOAD)

MULTI-PROTOCOL COMMUNICATIONS CONTROLLER



TYPICAL CONFIGURATION



MOTOROLA

SEMICONDUCTORS

3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721

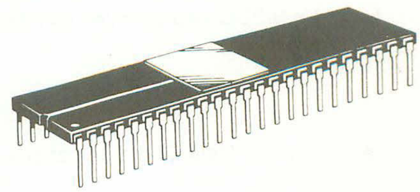
MC68340

Product Description

HMOS

(HIGH DENSITY N-CHANNEL, SILICON-GATE DEPLETION LOAD)

DUAL-PORT RAM

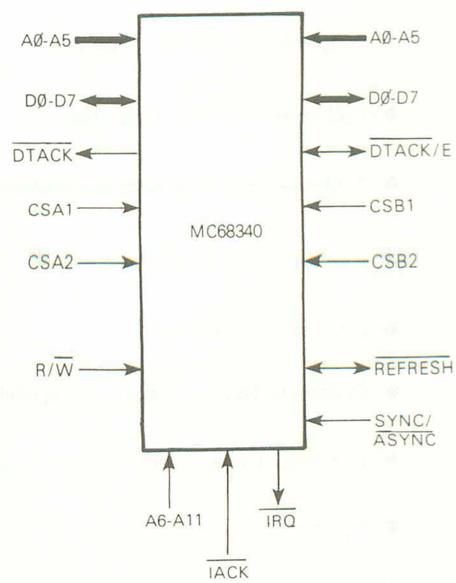


DUAL-PORT RAM

The MC68340 Dual-Port RAM (DPR) is intended for use as the primary data and control communications path between two processing elements in multi-processing systems. Other applications such as a CRT refresh or disk interface buffer will also find the DPR valuable.

- Compatible with both M6800 and M68000 Families
- Bus Compatible with other Processors having a HOLD or ACKNOWLEDGE Line
- 2K X 8 to 4K X 8 RAM — Expandable to any Width and Depth
- Pseudo-Static — Contains Internal Refresh Cycles
- Multiple Semaphores
- Supports Vectored Interrupts

FUNCTIONAL DIAGRAM



MOTOROLA

SEMICONDUCTORS

3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721

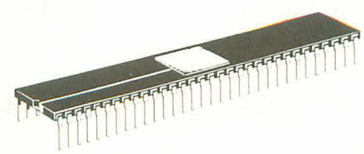
MC68560

Product Description

HMOS

(HIGH DENSITY N-CHANNEL, SILICON-GATE DEPLETION LOAD)

SERIAL DMA PROCESSOR



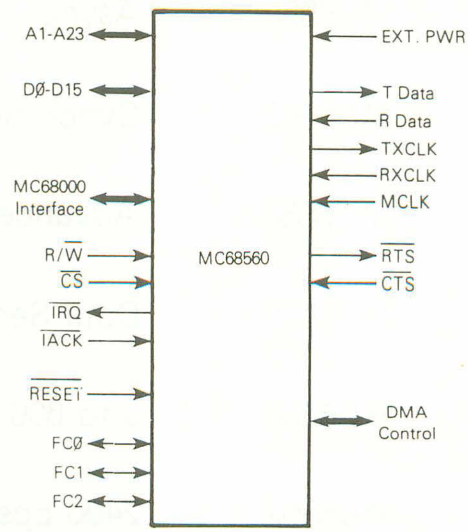
SERIAL DIRECT MEMORY ACCESS PROCESSOR

The MC68560 Serial Direct Memory Access (SDMA) processor provides a high speed serial link between microprocessors or intelligent controllers in distributed processing systems. Using IBM's Synchronous Data Link Control (SDLC) protocol, the MC68560 is capable of handling multidrop, point-to-point, or loop configurations in full or half duplex environments.

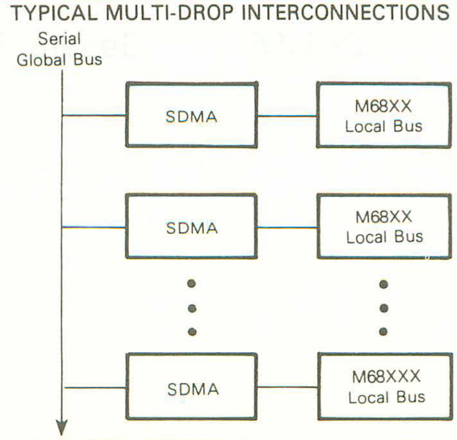
The serial link is configured with a primary SDMA connected to one or more secondary SDMAs. The primary station (SDMA and its associated processor) has the responsibility for the data link control, such as polling the secondary SDMAs. The secondary SDMAs have the ability to respond to most link-level commands received over the serial bus and also handle some error recovery without intervention from the local processor. Both the primary and secondary SDMAs have the responsibility for data integrity and validation.

- High Speed Serial Link
- External Data Recovery
- DMA Command and Data Chaining Capabilities
- NRZ Data
- Full or Half Duplex
- Supports SDLC and HDLC Protocols
- Internal Byte Synchronization
- External Clocks
- Point-to-Point, Multidrop, and Loop Modes Supported
- 24-Bit Address Bus; 16-Bit Data Bus
- MC68000 Bus Compatible
- 4 Megabit Serial Transfer Rate

FUNCTIONAL DIAGRAM

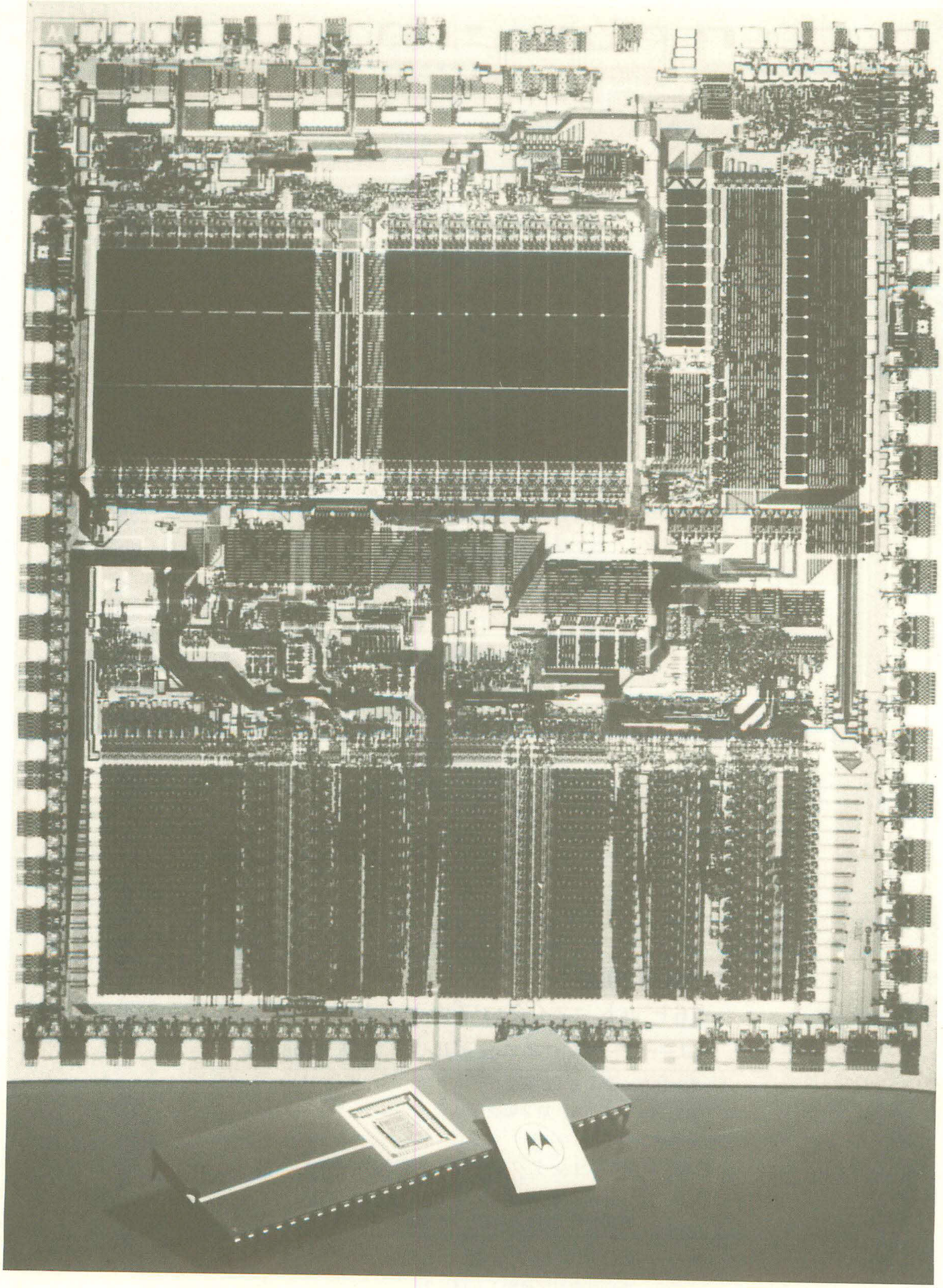


TYPICAL MULTI-DROP INTERCONNECTIONS



M6800 FAMILY PERIPHERALS THAT ARE DIRECTLY COMPATIBLE WITH THE MC68000

MCM6810	128 X 8 Bit Static RAM
MC6821	Peripheral Interface Adapter
MC6840	Programmable Timer Module
MC6843	Floppy Disk Controller
MC6845	CRT Controller
MC6847	Video Display Generator
MC6850	Asynchronous Communication Interface Adapter
MC6852	Synchronous Serial Data Adapter
MC6854	Advanced Data Link Controller
MC6859	Data Security Device
MC6860	0 to 600 bps Digital Modem
MC6862	2400 bps Modulator
MC68488	General Purpose Interface Adapter



THE MC68000 MICROPROCESSING UNIT

MC68000
16ビット　マイクロプロセッサ
ユーザーズ・マニュアル　　**定価　2,000円**

昭和56年６月１日　初版発行
昭和56年７月20日　第２版発行

発行所　モトローラ
セミコンダクターズ・ジャパン　株式会社
東京都港区南麻布５－２－32（第32興和ビル）
〒106　☎03（440）3311（大代表）

発売元　CQ出版株式会社
東京都豊島区巣鴨１－14－２
〒170　☎03（947）6311（代表）　振替　東京０－10665

定価 2,000円